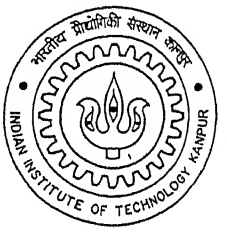
Evaluation of Pruning Algorithms

A Thesis Submitted
in partial fulfillment of the Requirement
for the degree of

MASTER OF TECHNOLOGY

by

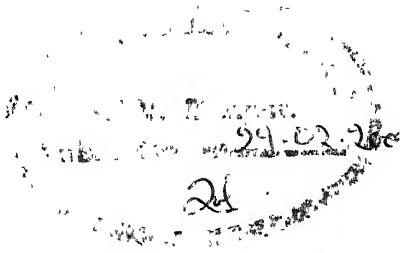
Tanmay Kumar Mishra



DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLGOY, KANPUR

February 2000

1 MAY 2000 EE
CENTRAL LIBRARY
I. I. T., KANPUR
A 130797
TH
EE/2000/M687



Certificate

It is to certify that, the work contained in the thesis entitled "**Evaluation Pruning Algorithms**" by Tanmay kumar Mishra has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

(Dr.P.K.Kalra)
Professor
Department of Electrical Engineering
Indian Institute of Technology, Kanpur

DEDICATED
TO
MY PARENTS

ABSTRACT

An intelligent system is excepted to take its own decisions taking into consideration firstly, past experience and secondly, the knowledge acquired by a common man regarding the operation of the system. To achieve this objective, **neural network** & **fuzzy logic** were developed. Neural network imitates the functions of the smallest biological entity of human brain, **the neuron**. It can be used for prediction of future, classifying objects, storing the information like a memory etc. Fuzzy logic gives a mathematical shape to human inference of a situation. These methods can be realized by computer simulations very easily. But, for real life applications, artificial neural network, fuzzy logic need to be implemented as analog, digital or hybrid (analog/digital) hardware.

One of the most important features of artificial neural systems is that they perform a large number of numerical operations in parallel. These operations involve, among others, simple operations as well as nonlinear mappings and computation of derivatives etc. . It is very difficult to predict actual number of nodes, connections to start with. This is because small number of weights may not be able map input-output relationship exactly and on the other hand, high number of weights increase the hardware cost. To optimize the weight connections and number of nodes without affecting the network's generalization capability, researchers suggested network growing and network pruning algorithms. Network pruning method is proved to superior than network growing method in literature. The former aims to delete the excess weight connections, hidden nodes, input variables which are having less important in network's input-output behavior.

One of the most popular **neural network** model, is the multi-layer feed-forward network using back propagation algorithm. Due to supervised nature of the learning method, it has been applied to various fields such as data compression, image processing and speech recognition. In this thesis, an attempt has been made to implement various pruning algorithms available in literature with the feed-forward back propagation network and compare the results. Results with different activation functions, different learning rates, network architecture are tried to get optimum solutions. Some of the input pruning methods are also reviewed in this thesis.

ACKNOWLEDGEMENT

I would like to express my gratitude to Dr. P.K. Kalra, my thesis supervisor, for his inspiring guidance and sincere supervision not only during the thesis work, but also during the complete academic program both inside and outside the lab. I am thankful to him for the complete freedom afforded to me in my work as well as for the excellent facilities in the lab. The completely open atmosphere for exchanging views in the lab, not only with him but also with all the others, was what helped me in achieving what I could, in this short span of time.

I would also like to express my special thanks to Manoranjan Sinha for his invaluable ideas and the time he spared from his Ph.D thesis work to help me in reviewing the results and my work.

My thanks to Prasanth, Madhav, Maj. Sidana, Murli mohan, T Murli krisna, Prasad, Sudheer, Tulasi, Raja for their words of encouragement, co-operation during the total period of stay here.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER-1

INTRODUCTION

The aim of modern technology is to built an expert system which will take care of every situation such as automatic decision making basing on past experience, classifying data into different clusters, implementing common man's experience regarding operation of the system. To fulfil this task, Neural Network and Fuzzy Logic were developed in last decade. Fuzzy Logic converts the common man's inference about a particular situation to a mathematical form and Neural Network gives the method for predicting the behavior of the system in future, classifying data into different clusters etc.

The origin of Neural Network comes from the smallest biological entity of human brain, the **neuron**. The characteristics a neuron forms the basis of Neural Network which is responsible for all types activities such as remembering the past for days together, apprehending the future events, all types arithmetic calculations, distinguishing something from a group etc. . Some of the well known applications of Neural Network are prediction of future trend in the field of finance, power sector, classification of data, pattern recognition, functional approximation.

The general architecture of Neural Network model contains a large number of artificial neurons(nodes), weight connections for performing different

operations. The exact number of hidden nodes, weight connections is very difficult to formulate. Researchers have indicated that less number of neurons result in inappropriate modeling of input-output On the other hand, large number of neurons results in increased hardware implementation cost and complexity in manufacturing. Similarly modeling of multi-input systems e g. chemical processes, metallurgical processes require a large number of inputs. Some of these have less contribution towards the input-output behavior of the system. In this context it is very important to formulize methods which could give the optimum number of hidden neurons, weight connections and input variables needed.

In literature, two solutions to the problem of minimum weight connections have been proposed, one is **network growing** and the second one is **network pruning**. *Network growing* method suggests adding neurons to the network to fulfil the network goal with starting from a small sized network. The cascade-correlation learning architecture by Fahman and Lebierre (1990), structure-level adaptation by Lee et al. (1990) are the examples of network growing method. In the first method, at the time of starting, there is no hidden node and they are added to network one by one during training process until satisfactory performance is attained. Each new hidden neuron receives a synaptic connection from each of the input nodes and from each pre-existing hidden neurons. When a new hidden neuron is added, only the synaptic weights on the output side are trained repeatedly. In the second method, the structure of the network is adapted by changing the number of neurons and structural relationship among the neurons in the network In this method, when the estimation error after convergence is larger than a desired value, a new neuron is added to the network in a position determined by the learning behavior of the network. On the other hand, *network pruning* method suggests deleting the weights, which have less importance in network starting from a large sized network.

Hence the pruning methods can be broadly classified as **weight pruning(network pruning)** and **input pruning**. The term **input pruning** may used for the methods that suggest removal of input variables whose contribution is very much less as compared to other variables towards the input-output behavior of the network It may

also be viewed as the dimensionality reduction method as it reduces the dimensions of input space by removing the less important variables.

- INPUT PRUNING —

A complex system contains a number of inputs and outputs .To build an intelligent system which resembles such a system, a large number of input variables are required. But higher dimensionality of input space implies greater computational complexity. Input pruning may be viewed as removal of some of input variables, which have less influence on the input-output behavior of the network. Various methods available for this purpose can be divided into two major groups. They are

- Principal Component Analysis(PCA)
- Independent Component Analysis(ICA)

- WEIGHT PRUNING—

All the algorithms that propose different methods for deletion the excess weight connections, hidden nodes and give a minimized structure are included in this category. The available algorithms till date can be broadly classified into three major categories -

- Penalty function methods
- Sensitivity analysis methods
- Iterative pruning method

From the different architectures available in Artificial Neural Network(ANN), multi-layer perceptron is the simplest feed-forward architecture, which has the ability to solve some difficult and diverse problems by training them in a supervised manner. The popular algorithm in this type of architectures is back propagation algorithm. This algorithm is based on error-correction learning rule. It may be viewed as a generalization of an adaptive filtering algorithm i.e. Least Mean Square

(LMS) algorithm for single neuron model. It is first proposed by Werbos(1974) in Ph.D thesis and subsequently rediscovered by Rumelhart, Hinton, and Williams(1986).

❑ BACK PROPAGATION ALGORITHIM : A REVIEW

Basically the error back propagation process consist of two passes through the different layers of the network: a forward pass and a backward pass. In forward pass, an activity pattern (input vector) is applied to the sensory nodes of network and its effect propagates through network layer by layer. Finally a set of outputs is produced as the actual response of the network. During froward pass the synaptic weights of the network are all fixed. During the backward pass the synaptic weights are all adjusted in accordance with error correction rule. Specifically the actual response of the network is subtracted from a desired response to produce an error signal and this error signal is then propagated backward through the network. The back propagation algorithm performs an approximation to the global minimization achieved by the method of steepest-descent or by conjugate-gradient methods.

The general architecture of multi-layer perceptron may be shown as below(Fig.1.1).

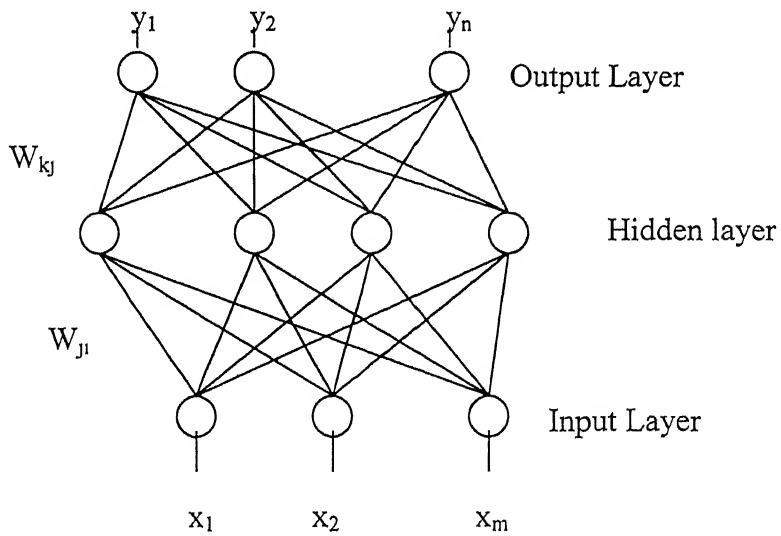


Fig.1.1 : Multilayer Perceptron

The input-output relationship of such a unit is characterized by the nonlinear recursive difference equation.

$$x_i^{(l+1)} = f\left(\sum_{j=1}^{N} w_{ij}^{(l)} x_j^{(l)} + \theta_i^{(l)}\right) \tag{1.1}$$

and this relation is generalized to all nodes in multi-layer perceptron as listed in Fig 1.

The error signal e_j required for adaptation is defined as the difference between the desired response and the output of the perceptron.

$$e_j(n) = d_j(n) - y_j(n) \qquad j = 1,2,.....N_M \tag{1.2}$$

Where $d_j(n)$ is the desired response at the j^{th} node of the output layer at time n ; $y_j(n)$ is the output at the j^{th} node of the output layer, and N_M is the number of nodes in the M^{th} layer or the output layer.

Hence the sum of the error squares produced by the network is

$$E(n) = \sum_{j=1}^{N_M} e_j(n) e_j^*(n) \tag{1.3}$$

The BP algorithm minimizes the cost functional *E(n)* by recursively altering the coefficient $\{w_{ij}^{(l)}, \theta_i^{(l)}\}$ based on the gradient search technique. Thus, finding the gradient vector of *E(n)* is the main idea of deriving the BP algorithm. The derivation of the BP Algorithm for second order and first order gradient-descent method is given at Appendix "A". The partial derivations of *E(n)* with respect to the coefficients of the output layer are first found at the output layer and they are then extended to the coefficients of all hidden units, thus arriving at the global minimum. The back-propagation networks (BPN) have been widely applied to data compression, speech processing, functional approximation, pattern recognition, classification etc.

- PROBLEM DEFINITION

The main aim of the thesis is

(a) To investigate the various pruning algorithms and functions available in literature in the feed forward network with Real-valued back propagation algorithm

(b) Compare the results of different activation functions for given Neural Network pruning algorithm.

- THESIS ORGANISATION

In this thesis different weight pruning algorithms are discussed. Brief review of different input pruning algorithms which are covered in the thesis entitled "**Dimensionality Reduction Methods in Intelligent System**" are included in chapter 2. Different weight pruning algorithms incase of single layer hidden network are discussed in Chapter-3. Chapter-4 contains the results of the various test problems tried with different activation functions and various optimization methods. Finally conclusion and future scope of the work is discussed in Chapter-5.

CHAPTER-2

INPUT PRUNING-A REVIEW

In many modeling problems, we often encounter the problem of having large number of inputs In available data, many input variables are considered due to unavailability of proper knowledge regarding the percentage of contribution of each variable in system's behavior. The large number of input variables results in a large sized network, which in turn indicates the increase in cost and complexity in the hardware implementation. Therefore it is very important to reduce the number of input variables by a method which determines the percentage of contribution of each input variable in system's input-output behavior. These methods are known as **dimensionality reduction** techniques. They may also be termed as **input pruning** methods as these methods aim to reduce the number of input variables. These methods are —

(a) Principal Component Analysis(PCA)

(b) Independent Component Analysis(ICA)

- PRINCIPAL COMPONENT ANALYSIS—

Principal component analysis (PCA) is the oldest and the best-known technique in multivariate analysis. It was first introduced by Pearson (1901), who used it in biological context to recast linear regression analysis into a new form. It was then developed by Hotelling (1933) in work done in psychometric.

Algebraically, principle components are particular linear combinations of uncorrelated p random variables $X_1, X2, ..., Xp$. Geometrically, these linear combinations represent the selection of a new coordinate system obtained by rotating the original system wth $X_1, X_2, ..., X_p$ as the coordinate axes. The new axes represent the directions with maximum variability and provide a simpler and more parsimonious description of the covariance structure. Hence summing up, the central idea of principal component analysis (PCA) is to reduce the dimensionality of a data set which consists of a large number of interrelated variables, to uncorrelated data set using second order statistics(covariance) of data set while retaining as much as variation present in it.

Principal components depend solely on the covariance matrix Σ (or correlation matrix ρ) of $X_1, X_2, ..., X_p$ .Let the random vector $\mathbf{X}'=[X_1, X_2, . , X_p]$ have covariance matrix Σ with eigen values $\lambda_1 \geq \lambda_2 \geq .. . \geq \lambda_p \geq 0$.

Let us consider the linear combinations

$$\mathbf{Y}_1 = l'_1\mathbf{X} = l_{11}X_1 + l_{21}X_2 + \ldots + l_{p1}X_p$$

$$\mathbf{Y}_2 = l'_2\mathbf{X} = l_{12}X_1 + l_{22}X_2 + \ldots + l_{p2}X_p$$

$$\mathbf{Y}_p = l'_p\mathbf{X} = l_{1p}X_1 + l_{2p}X_2 + \ldots + l_{pp}X_p$$

Hence

$$\mathrm{Var}(\mathbf{Y}_i) = l'_i \Sigma l_i \qquad i = 1,2,\ldots,p$$

$$\mathrm{Cov}(\mathbf{Y}_i, \mathbf{Y}_k) = l'_i \Sigma l_k \qquad i, k = 1,2,\ldots,p$$

The principal components are those *uncorrelated* linear combinations of $Y_1, Y_2, \ldots, Y_p$ whose variances are large as possible.

Therefore we can define

First principal component = linear combination $l'_1\mathbf{X}$ that maximizes $\mathrm{Var}(l'_1\mathbf{X})$ subject to $l'_1 l_1 = 1$

Second principal component = linear combination $l'_2\mathbf{X}$ that maximizes $\mathrm{Var}(l'_2\mathbf{X})$ subject to $l'_2 l'_2 = 1$ and $\mathrm{Cov}(l'_1\mathbf{X}, l'_2\mathbf{X}) = 0$

ith principal component = linear combination $l_i' \mathbf{X}$ that maximizes $\text{Var}(l_i' \mathbf{X})$

subject to $l_i' l_i' = 1$ and $\text{Cov}(l_i' \mathbf{X}, l_k' \mathbf{X}) = 0$ for $k<i$

Suppose V matrix represents the eigen vector matrix of the covariance matrix.

If $\lambda_1, \lambda_2, \ldots, \lambda_p$ be the eigen values of corresponding eigen vectors.

Then the proportion of total variance due to the kth principal component is

$$\begin{pmatrix}\text{Propotion of total population} \\ \text{variance due to } k\text{th princpal component}\end{pmatrix} = \frac{\lambda_k}{\lambda_1 + \lambda_2 + \ldots\ldots + \lambda_p} \qquad k = 1,2,\ldots\ldots,p$$

Hence the PCA analysis algorithm can be divided in the following steps—

Step-1—The input data set is normalized to have zero mean and standard deviation of unity.

Step-2—The covariance matrix is then calculated and the eigen vector and the eigen values of covariance matrix is calculated. The percentage loading of each eigen value is determined.

Step-3—According to the percentage of loading of each eigen value, the amount of input feature space to be considered is decided. The principal components are found by multiplying the transpose of eigen vector matrix with the input data set. By taking that decided number of variables from that principal component matrix, we get reduced input data set.

- INDEPENDENT COMPONENT ANALYSIS—

Independent Component Analysis (ICA) of a random vector consists of a linear transformation that minimizes the statistical dependence between its components. It may be considered to be an extension of Principal Component Analysis (PCA). In PCA, the data set are expressed as a linear combination of data vectors which are uncorrelated, but in ICA they are mutually independent. It is often considered as

"nonlinear PCA" method. It involves higher order statistics, which is essential for describing some of fundamental characteristics of non-Gaussian type data set.

The algorithm for ICA consist of three steps—

Step-1— This step is known as *whitening of the data set*. The data vectors x_k are made zero mean and they are made uncorrelated using Principal Component Analysis which also reduces the dimensions also.. The PCA whitening matrix is given by

$$V = D^{\frac{1}{2}} E^T$$

where $D = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_m)$, $E = (c_1, c_2, \ldots, c_m)$

λ_i denotes the largest eigen value of the correlaton matrix of given data set and c_i is the respective eigen vector.

The whitened vectors is given by $\mathbf{v_k} = \mathbf{V x_k}$

Step-2—The separation matrix **B** is learned with a feed forward back-propagation algorithm with a single layer of sigmoid transfer function. The learning rule is given by

$$\mathbf{W_{new}} = \mathbf{W_{old}} + \mu_{old} [\, \mathbf{v_{old}} - \mathbf{W_{old}}\, \mathbf{y_{old}} \,]\, \mathbf{y_{old}}'$$

where $\mathbf{W_{old}}$ is the previous and $\mathbf{W_{new}}$ is the updated weights and μ_{old} the learning rate and $\mathbf{y_{old}}$ is output of the neuron

Step-3—The last step consist of calculation of basis vector A. This vector is given by the formula

$$A = B^T (BB^T)^{-1} = ED^{\frac{1}{2}} W$$

This vector is used for selecting the variables in PCA i.e. associate one variable with each of basis vectors, namely the variables not already chosen with highest coefficient, in absolute value in each successive basis vector.

CHAPTER-3
WEIGHT PRUNING

- NECESSITY OF PRUNING METHOD—

Literally the word "**pruning**" means deletion of dead or overgrown parts. Hence in case of Artificial Neural Network(ANN), it is the deletion of unnecessary weights from the neural network .

In back-propagation learning, we use back propagation algorithm to compute the synaptic weights of a multi-layer perceptron by loading as many of the training examples as possible into ANN. An ANN is said to generalize well when the input-output relationship computed by the network is accurate for input-output patterns (test data) never used in creating or training the network. It is assumed that the test data are drawn from the same population used to generate the training data.

The learning process (i.e. training of a neural network) may be viewed as a "**curve fitting**" problem. Hence the ANN performs useful interpolation because multi layer perceptrons with continuous activation functions lead to output functions that are also continuous. When an ANN is trained for a large number of input-output examples, it may fit a higher order polynomial between the input-output examples. On the other hand, when it is trained for a few numbers of input-output examples, the problem of underfiting may arise. Again, from the architectural point of view, it is evident that the learning process of ANN depends on the number of hidden nodes.

Hence the generalization capability of ANN, i.e. capability of mapping accurately the test patterns not used for training is influenced by the following three parameters (Hykin):

a) The size of the training set

b) The architecture of the ANN

c) The physical complexity of the problem at hand

The results derived by Baum and Haussler (1989) indicates that the number of training examples required is directly proportional to the nuber of synaptic weights in the ANN and inversely proportional to the accuracy parameter $\in$. Thus with an error of 10 percent, the number of training examples required is approximately 10 times the number of synaptic weights in the network. So for actual practice, it is very difficult to choose training set which will solve the generalization problems.

To overcome this problem, the method of **cross-validation** may be used (Stone, 1974;Janssen et. al 1988). In this method, the available data set is randomly partitioned into :

1. (a) A subset used for estimation of the model(i.e. training the network)
   (b) A subset used for evaluation of the performance of the model (i.e. validation); the validation subset is typically 10 to 20 percent of training set.
2. A subset to test the generalization capability of ANN and it is not used for training.

The underlying idea is to validate the model on a data set different from the one used for parameter estimation. In this method, the training set is used to access the performance of the various ANN structures with different number of hidden units and the architecture with the best performance is chosen as the minimal correct architecture. This method also suggests stopping the training procedure when the validation error increases though training error decrease. This increase in error is due to the irregularities in sampling (by Chauvin, 1990).

This method of cross-validation, however, may not be practical when only a small amount of data is available. It is due to the fact that, validation data cannot be used for training and hence the size of training set is reduced. It is rather impossible to

have a perfectly trained ANN with a small number of training data.. Though it suggests a method of finding the actual number of hidden neurons by searching the architecture which will provide best performance, it is not practically possible due to the hit and trial nature of the process.

Another way of avoiding over-training is to limit the ability of the network to take advantage of spurious correlation in the data. Overfitting is thought to happen when the network has more degrees of freedom (weights, biases etc.) than the number of the training samples. Again to solve the real world problems with ANN, we usually require the use of highly structured networks of a rather large size. This is because of the fact that, the large sized network has the benefit of learning quickly due to the non-linearity involved. A practical issue that arises in this context is to construct a network with minimum size, which is less likely to learn noise in training data and may thus provide a good generalization.

To solve the above practical issue of exact number of the hidden neurons and weights, researchers have proposed **network pruning** methods. In these methods, we start with a huge sized network with adequate performance for problem at hand, and then prune it by weakening or reducing the redundant weights, nodes.

Hence we can conclude that, if the available data set is large enough to be divided into training set, validation set and the testing set, then we can divide the data set to above three sets and train the network properly to have an ANN with a good generalization ability. But if the data set is less, then it is better to choose a huge sized network to train the input-output behavior. After the training or during training remove the additional weights to get the optimal size of the network without affecting the generalization capability of the network.

In this thesis some the methods available for weight pruning are discussed. The main method available are-

1. Penalty function methods or Complexity regularization
2. Sensitivity analysis methods
3. Iterative pruning method

- COMPLEXITY REGULARIZATION OR PENALTY FUNCTION METHODS —

To design the network we need an appropriate measure of the fit between the model and the observed data. This implies that the design procedure should include a criterion for selecting the model complexity (i.e. the number of independently adjusted parameters of the network). Various criteria for model-complexity selection are described in the statistics literature, important examples of which are the *minimum description length* (MDL) criterion (Rissanen, 1978,1989) and an *information–theoretic* (AIC) (Akaike, 1974).

The common forms of composition of these methods are

$$\begin{pmatrix} Model-complexity \\ criterion \end{pmatrix} = \begin{pmatrix} \log-likehood \\ function \end{pmatrix} + \begin{pmatrix} \mathrm{mod}\,el-complexity \\ penalty \end{pmatrix}$$

In context of back-propagation learning, or any other supervised learning procedure the above relation can be viewed as to find the weight vector that minimizes the total risk($\boldsymbol{R}$) ,which is defined as

$$R(w) = E_s(w) + \lambda E_c(w)$$

The first term $E_s(w)$ is the **standard performance measure** which depends on both the network and the input data. In back-propagation learning, it is typically defined as sum-squared error whose evaluation extends over the output neurons

of the network and carried out for all training examples on an epoch-by-epoch basis. The second term $E_c(w)$ is the **complexity penalty**, which depends on the network and its evaluation extends over all the synaptic weight connections in the network. The term λ is called **regularization parameter**, which represent the relative importance of complexity-penalty term with respect to performance-measure term. When λ is zero, the back-propagation learning process is unconstrained, with the network is completely determined from the training examples. When λ is made infinitely large, on the other hand, the implication is that the constraint imposed by the complexity penalty is by itself sufficient to specify the network, in other words the training examples are unreliable.

There are basically three penalty function methods available

1. Weight decay method-1
2. Weight decay method-2
3. Weight elimination method

In all these penalty function methods, the training method seems to minimize the new error function, which consist of the original error function and the penalty term. Hence the error calculation and weight updating term include an additional term for calculating the effect of penalty term.

- WEIGHT DECAY METHOD-1—

This method is proposed by Ishikawa (1990). The complexity regularization is defined as the absolute value of the weight vector **w** in the network. The complexity regularization term is given as-

$$E_c(w) = \sum_{i \in C_{total}} |w_i|$$

where C_{total} indicates all synaptic weights of network

and hence the new error function becomes

$$E(w_i) = E_s(w\ ) + \lambda \sum_{i \in C_{total}} |w_i|$$

Therefore the weight updating part of algorithm contains the following part in addition to original weight updating part of the respective algorithms

$$\Delta w_r = -\lambda \ \text{sign}(w_{old})$$

where Δw_r is the change in weight due to regularization term.

The addition of complexity regularization part to the weight updating part suggests that if $\mathbf{w_{old}} > 0$ then weight is decremented by amount equals to the regularization parameter and if $\mathbf{w_{old}} < 0$ then it is to be incremented by that amount.

- WEIGHT DECAY METHOD-2—

This method is proposed by (Hinton, 1987). The complexity regularization is defined as the squared norm of the weight vector **w** in the network. It is given by

$$E_c(w) = \frac{1}{2}\|w\|^2 = \frac{1}{2} \sum_{i \in C_{total}} w_i^2$$

where C_{total} is all synaptic weights of the network

and hence the new error function becomes

$$E(w) = E_s(w) + \frac{\lambda}{2} \sum_{i \in C_{total}} w_i^2$$

Therefore, the weight updating part of the algorithm contains the following part in addition to the original part weight updating part as suggested in the respective algorithm

$$\Delta w_r = -\lambda(w_i)$$

where Δw_i is the change in weight due to regularization term

This procedure operates by forcing some of the synaptic weights in the ANN to take on values close to zero, while forcing other weights to retain their relatively large values.

Accordingly, all the synaptic weights of the ANN are grouped into two categories: those that have a large influence on the model, and those that have little or no influence on it. The weights in latter category are referred to as excess weights. In absence of complexity regularization, these weights result in poor generalization by virtue of a high likelihood of taking on completely arbitrary values or causing the ANN to overfit the data in order to produce a slight reduction in training error (Hush and Horne, 1993). The use of complexity regularization encourages the weights having less influence to assume values close to zero and thereby improve generalization.

- WEIGHT ELIMINATION—

This method is proposed by Weigend et al.,(1991). In this method the complexity regularization term is defined by

$$E_c(w)=\frac{1}{2}\sum_{i\in C_{total}}\frac{(w_i/w_0)^2}{1+(w_i/w_0)^2}$$

where $\mathbf{w_0}$ is a pre-assigned free parameter close to unity

C_{total} is all synaptic connections of the network

and hence the new error function becomes

$$E(w)=E(w_i)+\frac{\lambda}{2}\sum_{i\in C_{total}}\frac{(w_i/w_0)^2}{1+(w_i/w_0)^2}$$

Therefore the weight updating part of the algorithm contains the following part in addition to the original weight updating part of the respective algorithms

$$\Delta w_r=-\lambda\frac{w_i w_0^2}{\left(w_0^2+w_i^2\right)^2}$$

In this type of penalty function, when $|w_i| << \mathbf{w_0}$ the complexity penalty for that weight approaches zero. The implication of this condition is that the *i*th synaptic weight is unreliable and should therefore be eliminated from the ANN. On the other

hand, when $|w_i| >> w_0$, the complexity penalty for that weight approaches the maximum value of unity, which means that w_i is important to the back-propagation learning process.

It can be noted that weight elimination procedure includes weight decay procedure as a special case. The weight-elimination procedure is particularly sensitive to the choice of the regularization parameter λ.Weigend et al.(1991) describe a set of three heuristics for the incremental adjustment of λ.The procedure starts with λ=0,enabling the ANN to use all of its resources initially and then λ is adjusted by a small amount after the presentation of each pattern in epoch.

Procedure for change of the regularization parameter (λ)

Let n denote the iteration for particular epoch that has just finished.
Let R(n) denote the corresponding value of the total risk. Since R(n) can increase or decrease from one iteration to the next, it is compared to three quantities:

(a) previous risk, R(n-1)

(b) Average risk, defined by $R_{av}(n)=\gamma R_{av}(n-1)+(1-\gamma)R(n)$, where $0<\gamma<1$

(c) Desired risk, D

The first two quantities, R(n-1) and $R_{av}(n)$, are derived from previous values of the risk itself, whereas D is supplied externally. The choice of D depends on the problem at hand. Thus, given R(n-1),$R_{av}(n)$ and D, the risk R(n) is compared to these quantities and depending on the results of comparison, one of three possible actions is taken as follows:

1. Increment the regularization parameter λ by a fairly small amount Δλ and thereby adding slightly more importance to complexity penalty term.

   if **R(n)< D and/or R(n)<R(n-1)**

   The adjustment Δλ is typically of order 10^{-6}.
2. Decrement the regularization parameter λ by fairly small Δλ and thereby attach slightly more importance to the performance measure term.

   if **R(n) ≥R(n-1), R(n)<R_{av}(n) and R(n)≥ D**

3. Reduce the regularization parameterλ by 10 percent and thereby attaching much more importance to the performance measure term.

if $\mathbf{R(n) \geq R(n-1),\ R(n) \geq R_{av}(n)\ and\ R(n) \geq D}$

Salient features of penalty function methods—

1. This method suggests modification of the performance criterion of the ANN. It has no influence on the type of the training procedure followed.

2 It controls the amount of pruning by changing the regularization parameter and suggests that the optimal settings of the weights and regularization parameter is attained when a smooth error surface is attained with the constraint that the final converged error is within the specified limit.

3 This method does not give an efficient reduced ANN as it only reduces less important weights to much less but does not eliminate them.

- SENSITIVITY ANALYSIS METHOD —

In general, as the training proceeds the synaptic weights converge to obtain the minimum value of the error function. Hence the sensitivity of each weight may be a criteria for deletion of the unnecessary weights from the ANN. Mozer and Smolensky (1989) have introduced this idea of estimating the sensitivity of error function to the elimination of each unit.

This method is broadly classified into two categories :

- Sensitivity method
- Hessian matrix calculation method

- SENSITIVITY METHOD —

Karnin(1990) suggests this method that measures the sensitivity of error function with respect to the removal of each connection and prunes the weights with low sensitivity.

The sensitivity of weight $\mathbf{w}_{ij}$ is given as

$$\mathbf{S}_{ij} = -\frac{E(w^f) - E(0)}{w^f - 0} w^f$$

Where w^f is the final value of the weight after training,

0 is the initial value of synaptic weights

$E(0)$ is the initial value of error

$E(w^f)$ is the final value of error after training

Since the weights are initialized at w^i instead of origin, the above formulae is modified to the following

$$\mathbf{S}_{ij} = -\frac{E(w^f) - E(w^i)}{w^f - w^i} w^f$$

Due to discrete nature of training procedure, the error surface can be approximated as

$$E(w^f) - E(w^i) = \int_i^f \frac{\partial E(w)}{\partial w} dw = \sum_{n=1}^{N} \frac{\partial E(w)}{\partial w} \Delta w_{ij}(n)$$

where i = initial state , f = final state, N= total number of epochs of training

Thus the estimated sensitivity of weight connection w_{ij} is given by

$$\mathbf{S}_{ij} = -\sum_{n=0}^{N-1} \frac{\partial E}{\partial w_{ij}} \Delta w_{ij}(n) \frac{w_{ij}^f}{w_{ij}^f - w_{ij}^i}$$

So in this method of pruning procedure, at the end of training process, the sensitivity of each weight is calculated and the lowest sensitivity weights are deleted. If all output connections from a node are deleted, the node itself is deleted. If all the input weights to a node are deleted, it will have constant output and can be deleted after adjusting for its effect on the bias of the following nodes. After this pruning phase the network is retrained to adjust the weights to compensate the increase in error. This pruning process is continued till abrupt change in sensitivities of the weights is observed.

➢ HESSIAN MATRIX CALCULATION METHOD —

This method is developed by Le Cun et al.(1990). This method is also known as "**optimal brain surgeon**". This method is based on the measurement of the "saliency" of a synaptic weight by estimating the second derivative of the error with respect to the weight. The basic idea of this pruning approach is to make use of information on second order derivatives of the error surface in order to make a trade-off between network complexity and training-error performance. The local approximation of the cost surface using a Taylor series about the operating point is given by

$$\partial E = \sum_i g_i \delta w_i(n) + \frac{1}{2}\sum_i \sum_j h_{ij} \delta w_i \delta w_j + \text{ higher order terms}$$

where δw_i denote the perturbation in synaptic weight w_i .

$g_i = \frac{\partial E}{\partial w_i}$ is the ith component of the gradient vector of E

$h_{ji} = \frac{\partial^2 E}{\partial w_j \partial w_i}$ is the jith component of the Hessian matrix of E

both measured with respect to the parameters of the network

The objective is to identify a set of parameters whose deletion from the multi layer perceptron will cause the least increase in the value in the cost function E. This calculation of Hessian matrix depends on the following assumptions—

a) External Approximation- The parameters are assumed to be deleted from the network only after the training process has converged. The implication of this assumption is that the parameters have a set of values corresponding to a local minimum or global minimum of the error surface. In that case the first term of the Taylor series expansion becomes zero.
b) Quadratic Approximation- The error surface around a local minimum or global minimum is assumed to be **quadratic**. This implies the deletion of higher order terms.

Therefore the approximated error surface will be

$$\delta E \cong \frac{1}{2}\sum_{i}\sum_{j} h_{ji}\delta w_i \delta w_j = \frac{1}{2}\delta \mathbf{w}^{\mathrm{T}}\mathbf{H}\delta \mathbf{w}$$

where $\delta\mathbf{w}$ is the perturbation applied to weight vector **w**, and

H is the Hessian matrix containing all second order derivatives to elements of the weight vector **w**

The objective of this method is to minimize the incremental increase in E as one of the synaptic weights to zero which can be achieved by solving in **Lagrangian** give below-

$$\mathrm{S} = \frac{1}{2}\delta \mathbf{w}^{\mathrm{T}}\mathbf{H}\delta \mathbf{w} + \lambda(\mathbf{1}_i^T \delta \mathbf{w} + w_i)$$

where $\mathbf{1}_i$ is the unit vector whose elements are all zero,except for the *i*th element which is unity and λ is *Lagrangian multiplier*

Therefore the optimum value of the **Lagrangian** S or the saliency of weight **w** is given by

$$S_i = \frac{1}{2}\frac{w_i^2}{[H^{-1}]_{ii}}$$

where $\mathbf{H}^{-1}$ is the inverse of the Hessian matrix **H** and $[\mathbf{H}^{-1}]_{ii}$ is the *ii*th element of this inverse matrix.

In effect, the saliency S_i represents the increase in the squared error (performance measure) that results from the deletion of $\mathbf{w}_i$. The Hessian matrix is always nonsingular in case of back-propagation learning. Here pruning is done iteratively, i.e. train to a reasonable error level, compute saliency, delete low saliency weights and resume training. This method can be simplified by making the assumption that the Hessian matrix is a diagonal matrix, which is known as "**optimal brain damage**".

Salient features of Sensitivity Analysis Method—

1. The sensitivity method deletes the lowest sensitive weight first and hence, there may be a natural stopping point where sensitivity jumps suddenly, which can be used as a pointer for stopping pruning.

2. The second order Hessian matrix method proves to be the best pruning method when the program for retraining is available as it takes to consideration of second order information of the error surface.
3. These methods suggest the retraining of the network, which will reduce the increased error due to the elimination of weights.
4. The only disadvantages with these methods are the amount of computation involved and the amount of memory storage space required and the necessity of retraining process.

- ITERATIVE PRUNING METHOD —

This method is proposed by Castellano and Fanelli (1997). This is a post-training pruning method for feed-forward network, which aims to select the optimal size by gradually reducing a large trained network. The method is based on the idea of iteratively removing hidden units and then adjusting the remaining weights with a view to maintaining the original input-output behavior. At each step of pruning, the net input of the units fed by the unit being removed is kept approximately constant before and after the removal of the unit, across the entire training set. This leads to a system of linear equations that we solve in least-squares sense using a preconditioned conjugate gradient procedure. The derivation of the method is discussed in Appendix B.

In this procedure, the hidden unit to be removed is chosen by finding the hidden unit, which gives the minimum output value over all training patterns. Then increase in the weights connecting to those units, which are in the projective field of the hidden unit to be removed, are calculated. After removal of that hidden node, the pruned network's performance (i.e. the increase in error) is calculated. The above procedure is continued till the remaining network's performance detoriates excessively. This method can also be used for removing the synaptic weight connections instead of hidden units like weight elimination method. For that, the weight connection having the least value over all training patterns are taken to consideration and all the calculations are done similarly as that while replacing the hidden units. If separate training and validation sets

are available, the network's performance can be measured as the error rate over validation data to improve generalization. This method requires very less number of computations than the OBS procedure.

Salient features of Iterative Pruning Method—

1. This method is free from the free parameter choice like penalty function methods. It only depends only one free parameter.
2. This method does not require any retraining phase after pruning like the sensitivity pruning procedures.
3. This method is free from the training procedures. Hence the method is free from the initial weights, the learning dynamics.
4. The iterative nature of the algorithm permits the network designer to monitor the behavior of the reduced networks at each stage of the pruning process, so as to define his own stopping criterion.

CHAPTER-4

COMPARISON OF RESULTS OF DIFFERENT WEIGHT PRUNING ALGORITHMS

All the pruning algorithms are applied to two different problems. The first problem is a functional approximation problem and the second one is the classical classification problem (XOR problem). This chapter contains the description of the test problems and their results obtained with different combinations of activation functions in hidden and output layer and with different number of hidden neurons in a single hidden layer network architectures. The comparison of these results indicates the difference between different algorithms. Again to compare the effect of pruning for each pattern at the end of each problem, table containing output values at each training pattern are given.

The function considered here is given by

$$f(x) = \sin x \sin y \qquad \text{where } x \in [0.1, 6.1] \text{ and } y \in [0.1, 6.1]$$

given in radians

This problem falls in a category of two-input-one-output problem. The data set consists of 49 training patterns. The objective of pruning algorithm is map input-output relationship as accurately done in case of an unpruned network. Since a perfectly trained network must able to map perfectly on the training set, to test the pruning algorithms the total data set is given as training set without dividing it into training set and validation set. The cost function is taken to be sum-squared error (SSE). Second order algorithm such as Levenberg-Marquardt training procedure of back-propagation algorithm is followed for training. The weights are initialized between $\left(\frac{-2.4}{F_i}, \frac{2.4}{F_i}\right)$, where F_i is the fan-in of the ith node for every layer. The single hidden layer architecture is chosen to solve the problem for its simplicity. The activation functions for both layers are chosen to be sigmoid. The inputs and outputs are normalized between the -1.0 to 1.0. The input data and the output data is presented in fig. 4.1 given below.

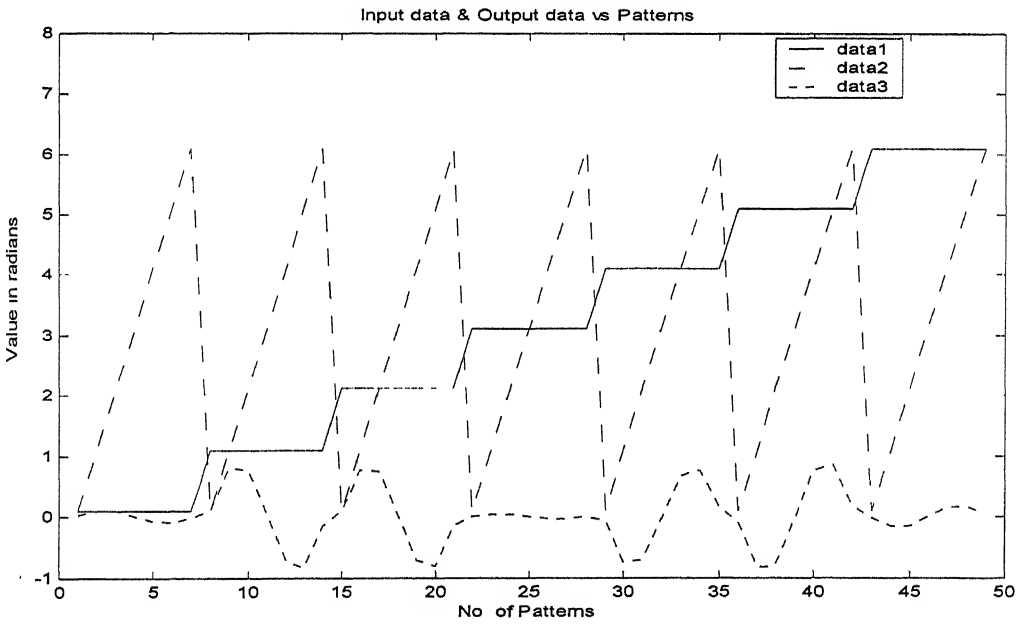


Fig. 4.1 Data set for training the problem

algorithms are described below –

Hessian matrix method-

This method is tried with two different architectures, which differs only by the number of hidden neurons and the steepness parameter of sigmoid unit (beta). The activation function used in both the cases is the sigmoid function. In first case the number of hidden neurons are 30 and in second case, it is 40.

The results with 30 hidden neurons are shown in fig.4 2 and the results are summarized in table 4.1 and table 4.2. The results with 40 hidden neurons are shown in fig.4.3 and the results are summarized in table 4.3 and table 4.4.

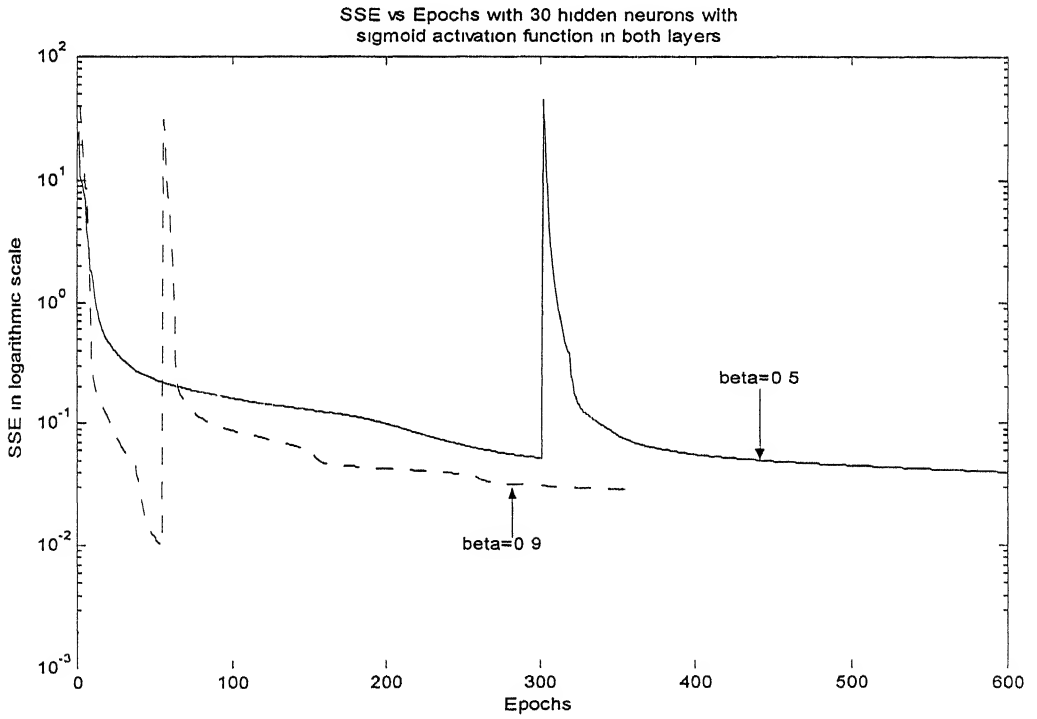


Fig. 4.2 Comparison of SSE curves using 30 hidden neurons with different values of steepness parameter of neuron (beta)

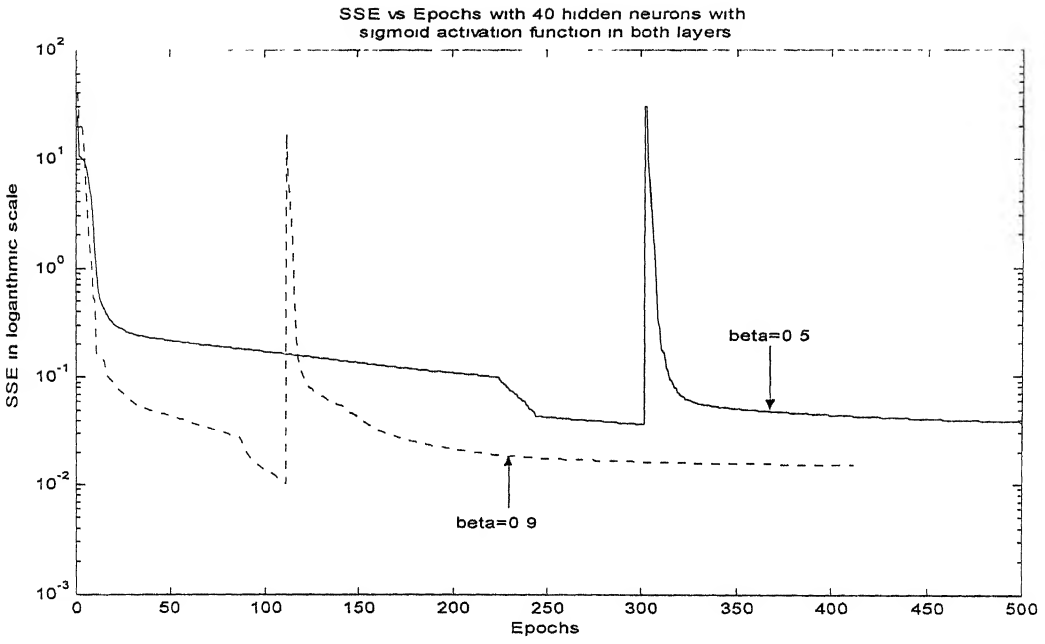


Fig.4.3 Comparison of SSE curves using 40 hidden neurons with different values of steepness parameter of neuron(beta)

Table no. 4.1 with beta=0 5, both layers have sigmoid activation

number of hidden neurons - 30
threshold value for deletion of weigths - 0.02
number of hidden units after pruning - 17
training epochs before pruning - 300
training epochs after pruning - 300
error goal for training process - 0 01

Table no. 4.2 with beta=0.9, both layers have sigmoid activation

number of hidden neurons - 30
threshold value for deletion of weigths - 0.02
number of hidden units after pruning - 10
training epochs before pruning - 55
training epochs after pruning - 300
error goal for training process - 0.01

Table no.4.3 with beta=0.9, both layers have sigmoid activation

number of hidden neurons - 40
threshold value for deletion of weigths - 0.02
number of hidden units after pruning - 16
training epochs before pruning - 110
training epochs after pruning - 300
error goal for training process - 0.01

Table no.4.4 with beta=0.5, both layers have sigmoid activation

number of hidden neurons - 40 threshold value for deletion of weigths - 0.02 number of hidden units after pruning - 25 training epochs before pruning - 300 training epochs after pruning - 300 error goal for training process - 0.01

Discussions-

The observed peaks are the increased error just after pruning and again the error value is reduced in the process of retraining Hence the benefits of this method can be easily pointed out as it automatically compensates the increase in error due to the pruning process. But the number of training epochs after the pruning should be kept as much as near to the training epochs before pruning. It can be easily observed that the value of steepness constant (beta) can affect the convergence during the training period and the number of hidden nodes to be deleted. The number of hidden neurons also plays an important role in deciding the minimum number of hidden nodes after pruning as it affects the convergence speed of training algorithm. In this problem, it is observed when linear activation function is taken at output layer keeping all other parameter constant, number of weight connections between input to hidden and hidden to output layer after deletion is different.

Iterative Pruning Procedure-

The network is trained with the same initial weights that of the Hessian matrix method. After training procedure, the pruning algorithm is applied to for the different values of the pre-conditioning parameter (omega) and it is allowed to proceed until the SSE value in the pruned ANN is within a specified limit. The results with different hidden neurons, omega are described in table 4.5 and table 4.6 .

Table no. 4.5 Results with 30 hidden units

number of hidden neurons - 30
steepness parameter for sigmoid unit - 0.9
for omega = 0.5 , the number of hidden neurons after pruning - 28

for omega = 1.5 , the number of hidden neurons after pruning - 28
steepness parameter for sigmoid unit - 0.5
for omega = 0.5 , the number of hidden neurons after pruning - 29

for omega = 1.5 , the number of hidden neurons after pruning - 26

Table no. 4.6 Results with 40 hidden units

number of hidden neurons - 40
steepness parameter for sigmoid unit - 0.9
for omega = 0.5 , the number of hidden neurons after pruning - 34

for omega = 1.5 , the number of hidden neurons after pruning - 34
steepness parameter for sigmoid unit - 0.5
for omega = 0.5 , the number of hidden neurons after pruning - 34

for omega = 1.5 , the number of hidden neurons after pruning - 33

Discussions-

This method only depends on the activation function of the neurons, not the training procedures. Though this method has the advantage of not requiring retraining process, but the results are poorer as compared to Hessian matrix method. The effect of steepness parameter (beta) and the pre-conditioning parameter (omega) can be visualized from the above tables. In general, higher the value of omega, higher number of hidden units are deleted. This pruning process shows a drastic change in SSE value during the process of pruning, which is the point to stop pruning.

Penalty function Procedures-

- **Weight decay method-1-**

This pruning process is tried with different regularization parameter (lamda) with both 30 and 40 hidden neuron architecture. The error surface with different regularization parameter is given in fig. 4.4 and fig. 4.5 for 30 and 40 hidden neurons respectively. The weight values after the training process are listed in table 4.7,4.8,4.9,4.10,4.11,4.12,4.13,4.14 give an illustration of the pruning algorithm In tables,w1 indicates the input to hidden layer weights,w2 indicates the hidden to output layer weights.

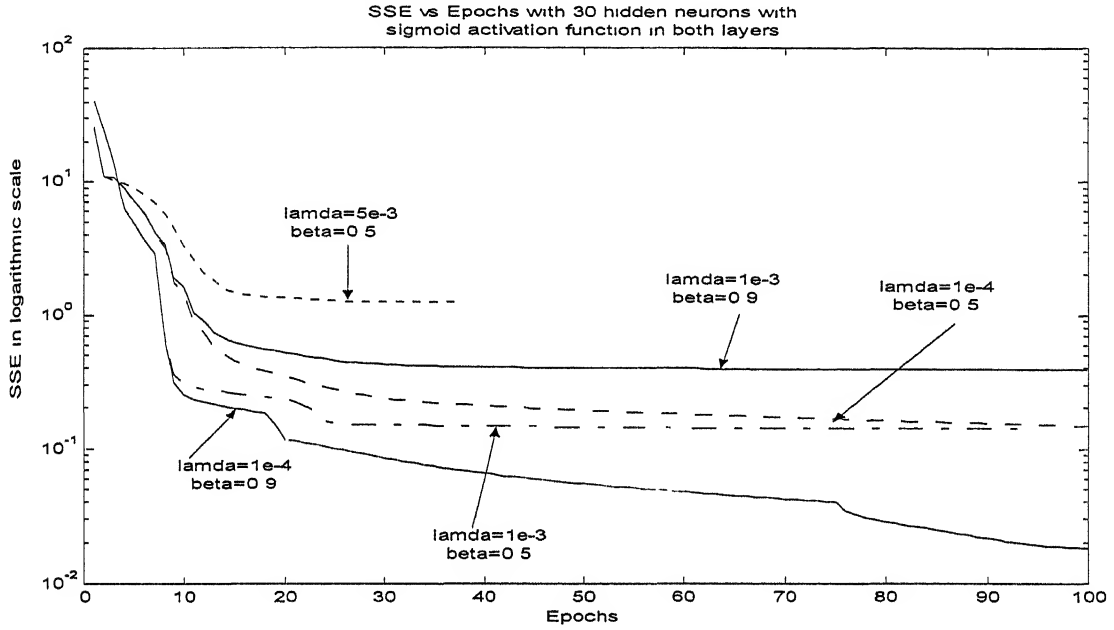


Fig. 4.4 –Comparison of SSE curves with different values of lamda

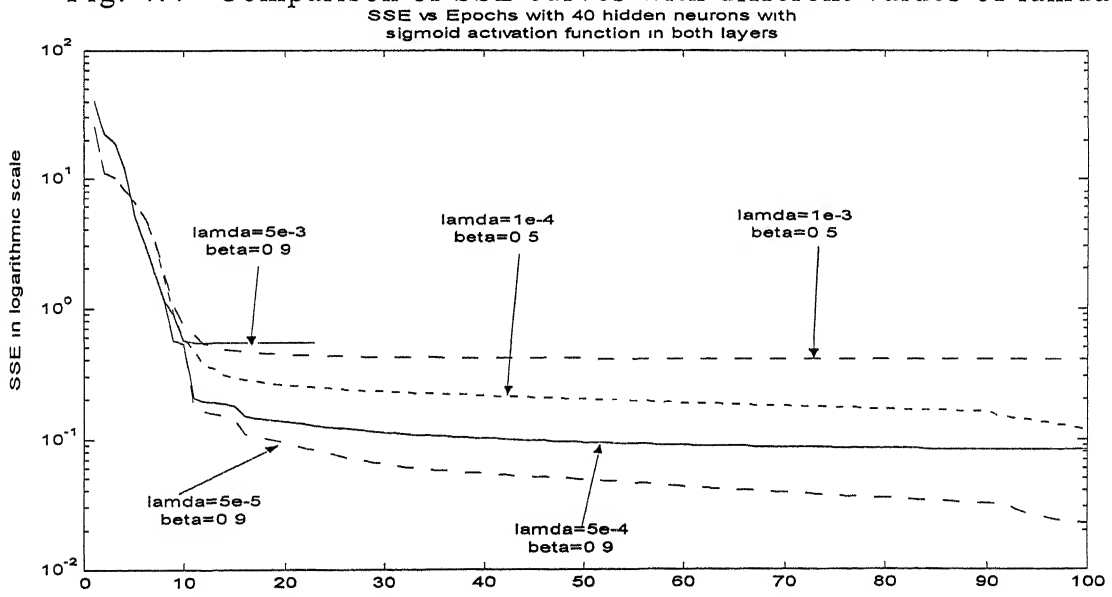


Fig 4.5.-Comparison of SSE curves with different values of lamda

For architectures with sigmoid(beta = 0.5) in both layers

Table no. 4.7 lamda=1e-4

w1		w2
0.3849	0.6920	0.4198
-4.6476	5.2594	5.3254
-1.2605	-0.1031	0.5926
0.0559	-0.0214	-0.4332
1.2557	0.5794	0.5927
2.4603	-1.2289	-2.2313
3.7834	-1.1474	-3.8760
-1.9113	2.2119	-3.6545
3.0266	2.7388	5.3565
-1.3313	2.6330	-2.7209
0.9807	2.9822	3.8796
-1.7721	2.2286	-3.4737
-2.1242	2.2355	-4.0663
-0.0410	-0.1285	-0.2982
-1.3532	-0.3351	0.9980
1.3606	-2.0485	-1.7795
-3.4928	-3.6530	-6.4356
2.1545	0.0994	0.6067
-1.5861	2.2620	-3.1208
-0.4413	-0.4423	0.2785
4.3984	-1.8391	3.3318
-1.9613	-0.9110	2.2168
2.1447	-1.4718	-2.1268
-1.7683	-1.1022	0.9474
2.3840	3 8906	-2.0646
-0.4996	-0.4643	0.3412
3.8812	3.0439	-3.9063
-4.7985	4.8961	4.4367
-2.6705	-0.9173	2.6128
3.4820	-4.1122	4.7900

Table no. 4.8 lamda=1e-3

w1		w2
0.4734	0.8446	0.6053
-4.4378	5.0896	5.1199
-1.3504	-0.1832	0.8112
0.1946	-0.1549	-0.5542
1.1549	0.9455	0.7314
2.0971	-1.1708	-2.2774
3.2142	-1.1667	-3.6236
-1.8698	2.1957	-3.4340
2.6603	2.5188	4.7828
-1.2700	2.3156	-2.5070
1.3127	2.6485	3.4936
-1.8061	2.1133	-3.3076
-1.9575	2.1269	-3 7357
-0.0877	-0.2282	-0.3637
-1.3275	-0.1479	1.0297
1.3508	-1.7266	-1.8557
-3.5257	-3.7153	-6.1705
2.2625	-0.0394	0.5004
-1.6314	2.2011	-3.0072
-0.3197	-0.3404	0.2665
4.1183	-1.8045	3.3755
-1.8036	-0.9050	2.2361
1.7153	-1.1906	-1.9689
-1.7530	-0.9581	1.1762
2.3989	3.4295	-1.9419
-0.3903	-0.3542	0.3481
3.9122	3.0266	-4.1101
-4.7471	4.8268	4.4297
-2.3480	-0.7892	2.3078
3.2156	-3.9186	4.5830

For architectures with sigmoid(beta=0.9) in both layers

Table no. 4. 9 lamda=1e-4

w1		w2
0.5833	0.5299	0.2271
-2.8870	2.7277	2.7199
-1.8519	-0.4938	1.5981
0.0620	0.1452	-0.3097
1.0067	-0.0068	0.1995
1.4281	-0.5868	-0.9094
2.6829	-2.5275	-3.1371
-2.9761	0.9646	-1.9570
1.7822	1.2222	1.8280
-0.3513	1.8781	-1.2955
2.2958	2.2603	2.4711
-0.9483	1.3118	-1.1067
-2.0196	1.8481	-2.4888
0.0577	-0.0966	0.0362
-2.6602	-0.8520	2.3337
1.1377	-2.9668	-1.2609
-2.8214	-2.7190	-3.6431
1.3699	0.1463	0.4683
-1.4566	1.2735	-1.6437
-0.0221	-0.6929	0.6568
1.3583	-1.5595	3.8633
-1.9048	-2.3546	3.1585
1.5697	-0.7945	-0.9789
-3.1014	-2.8055	-3.1787
0.3835	2.3329	-0.0573
-1.0118	-3.2094	2.0138
1.8147	1.6058	-2.6625
-3.1504	2.8504	3.2373
-1.5348	-1.1557	-0.1040
0.3750	-2.9436	0.6222

Table no. 4.10 lamda = 1e-3

w1		w2
0.6871	0.5988	0.2974
-2.1731	2.1415	2.5516
-1.9974	-0.7604	1.3970
0.0862	0.0261	-0.4713
0.9278	0.0747	0.0795
1.8213	-0.8542	-1.1046
2.2920	-1.8512	-2.2135
-1.9753	0.9348	-1.4299
1.6429	1.4816	1.6739
-0.4539	1.9865	-1.3572
1.8585	2.0318	2.1160
-1.0253	1.2442	-1.0876
-1.3733	1.5344	-2.0052
-0.0271	-0.1754	-0.1884
-2.2857	-0.8558	2.0785
0.8209	-1.7778	-0.8397
-2.0661	-1.8772	-3.4479
1.4293	0.0063	0.2868
-1.2585	1.4142	-1.5881
0.0946	-0.4610	0.3834
1.9059	-1.8619	3.6729
-1.5421	-1.6010	2.0318
1.4036	-0.9599	-0.9283
-2.9379	-2.0979	-2.2093
0.4058	2.2567	0.0011
-0.4129	-1.4336	0.9699
1.9616	1.5640	-2.8062
-2.4970	2.6465	2.6219
-1.5713	-1.1345	-0.2301
0.3172	-3.0093	0.5596

For architectures with sigmoid(beta=0.5)in both layers

Table 4.11 lamda=5e-4

w1		w2
1.5480	1.3847	-0.4288
-1.1230	-0 8078	-0.9381
0.1331	0.4378	0.2099
0.0994	-0.1981	-0.1622
2.5679	2.5295	-2.3385
0.8688	-0.0390	-0.2852
-0.5216	1.0269	-0.5960
-2.2477	-0.9110	-0.7855
1.5489	-0.5363	-0.7324
-2.3612	-1.8513	2.6513
0.5006	-1.3224	-0.5035
0.7259	1.9944	1.9212
1.7862	-1.7716	3.5464
0.7567	-0.1756	-0.2059
-2.1382	-1.9204	-3.0935
-0.8798	0.9806	-0.8963
1.7070	-0.5338	-0.8990
2.4337	2.4999	-2.5632
-0.1243	0.8023	0.1181
1.6441	0.5758	1.2760
-1.7810	0.3190	-1.0143
-0.6861	2.0909	-0.6405
1.4977	2.0246	2.3142
-1.4224	0.0594	-0.3609
-1.6349	1.9109	3.5277
-0.3795	2.1907	-1.4814
-1.4646	-1.2190	1.5173
0.5748	-1.4303	-0.7502
-2.8289	-1.1018	1.7619
-2.1306	0.3862	-1.4217
-2.5237	1.5585	1.6896
2.4059	-2.0284	-2.4895
0.6735	1.7474	0.8820
1.5067	0.5392	0.8030
0.2156	-0.1560	-0.1454
-0.5029	1.2479	-0.8723
0.9898	0.2788	0.2533
0.3087	-0.0554	-0.3748
-1.8377	1.5398	-2.0879
0.3770	0.1631	0.0683

Table 4.12 lamda=5e-3

w1		w2
1.6936	1.1280	-0.4049
-0.6043	-0.8875	-0.8131
0.3472	0.5526	0.5302
0.1192	-0.2247	-0.1473
2.1363	2.0348	-1.6387
0.7925	-0.0826	-0.2327
-0.5213	0.6748	-0.5107
-2.0982	-0.3860	-0.8698
1.0529	-0.6816	-0.5175
-1.2893	-1.3004	1.5007
0.4631	-1.0594	-0.5046
0.8615	1.3194	1.2344
2.0480	-2.0870	2.8223
0.6829	-0.2136	-0.1656
-1.9242	-2.0114	-2.8305
-0.8549	0.7688	-0.7438
1.2897	-0.3768	-0.4695
1.9074	2.1567	-2.0733
-0.1489	0.8063	0.0547
1.1358	0.7879	0.7901
-1.0442	0.1359	-0.2602
-0.6613	1.5799	0.0325
1.2349	1.3199	1.4069
-1.3214	-0.0204	0.0038
-1.9864	2.1831	2.6304
-0.8805	1.1816	-1.0850
-1.1972	-1.1173	1.3080
0.6297	-0.9281	-0.6913
-1.1916	-0.8885	1.0922
-1.0143	0.7153	-0.7684
-2.3632	0.8242	1.0474
1.4646	-1.2305	-1.4311
0.3738	1.2515	0.4925
1.0803	0.3983	0.3467
0.2115	-0.1514	-0.0911
-0.6846	0.8138	-0.7347
0.8687	0.2170	0.3257
0.3546	-0.1313	-0.3618
-0.8456	1.1771	-1.0978
0.3990	0.2265	0.2684

For architectures with sigmoid(beta=0.9) in both layers

Table no. 4.13 lamda=1e-4

w1		w2
4.5409	4.8015	-4.6244
-3.5111	-3.7915	-5.5490
0.0900	0.5321	0.5109
0.0707	-0.1960	-0.2459
3.4270	1.9122	1.6522
0.8021	0.0531	-0.1582
-0 5694	0.6025	-0.4946
-1.9634	0.0456	0.1973
3.2694	-1.0946	-3.4747
-3.8436	-3.5106	5.2956
-1 5759	-3.5583	2.5061
1.6345	3.2765	3.6416
3.4756	-3.1348	7.5485
0.6707	-0.0792	-0.0797
-4.5112	-1.6321	1.5961
-1.0671	1.4683	-1.2255
2.3279	-0.8727	-1.5761
4.4994	4.7716	-5.5486
-0.0167	0.9845	0.5430
1.0071	2.3940	2.3859
-2.8063	0.4753	-2.0325
0.5554	2.0297	1.1501
2.5921	3.0199	5.3071
-1.6855	0.2822	-0.5906
-2.7175	3.5402	4.9296
-0.8017	3.2986	-3.9644
-0.9915	-0.1980	0.1915
0.9744	-0.9708	-0.6945
-0.7077	-0.1692	0.2022
-1.2234	2.1836	-2.1359
-3.8371	3.1502	3.8981
2.8562	-3.3315	-4.8899
0.6506	1.6597	1.1661
2.0522	1.3615	2.3076
0.1878	-0.1585	-0.1536
-1.1657	1.0665	-0.9864
1.5268	0.6604	1.1645
0.3881	-0.1266	-0.3944
-2.8015	2.7943	-4.6104
0.4081	0.2687	0.3242

Table no. 4.14 lamda=1e-3

w1		w2
4.0130	4.1826	-3.8873
-3.3412	-3.9078	-5.1907
0.0259	0.5566	0 6821
-0.0238	-0.0858	-0.2397
2.6209	2.2474	-0.3758
0.8913	0.1749	-0.3355
-0.4879	0.7265	-0.8412
-1.8483	0.2771	0.4816
1.6783	-1.7723	-2.8176
-2.1962	-2.5053	3.9376
-1.2179	-2.2702	2.2678
1.3686	2.3616	2.7065
3.4143	-3.7374	5.9786
0.8500	-0.3094	-0.4564
-4.0643	-1.3364	-1.4511
-0.7810	1.2597	-1.4734
1.8779	-0.7429	-1.0923
3.7072	4.6318	-4.3295
-0.0436	0.7553	0.5763
1.2512	1.9260	2.1191
-1.3662	0.5196	-0 8898
0.2504	1.9235	0.7093
1.9600	2.6615	4.0027
-1.4547	0.0642	0.3098
-3.1691	4.2752	3.9918
-1.6924	2.4004	-3.4991
-0.6882	0.0191	0.9014
0.6124	-0.7587	-1.1334
-1.1575	-0.2745	1.0867
-1.5114	1.5297	-2.1967
-3.8670	2.6137	3.3035
1.8440	-1.8100	-3.3805
0.9753	1.6757	1.3887
1.3115	1.8250	2.0443
0.3445	-0.0520	-0.2187
-0.6057	0.9440	-1.0969
1.1708	0.5939	1.0287
0.6497	-0.1077	-0.6578
-1.0964	2.7912	-3.6987
0.4882	0.4193	0.5131

Discussions-

From the above weight matrix table, it is evident that this method does not produce any significant reduction in weight values. In the weight table, we find some of the weights are reduced to minimum i.e. less than 0.001 and some of the weights are saturated around 1.5 to 2.5. The error surface is very much sensitive to the regularization parameter (lamda). The increase in the value of lamda saturates the network and the network does not converge to desired minimum value.

- **Weight decay method –2**

This pruning process is tried with different regularization parameter (lamda) with both 30 and 40 hidden neuron architecture. The error surface with different regularization parameter is given in fig. 4.6 and fig. 4.7 for 30 and 40 hidden neurons respectively. The weight values after the training process are listed in table 4.15,4.16,4.17,4.18,4.19,4.20,4.21,4.22 give an illustration of the pruning algorithm. In tables,w1 indicates the input to hidden layer weights,w2 indicates the hidden to output layer weights.

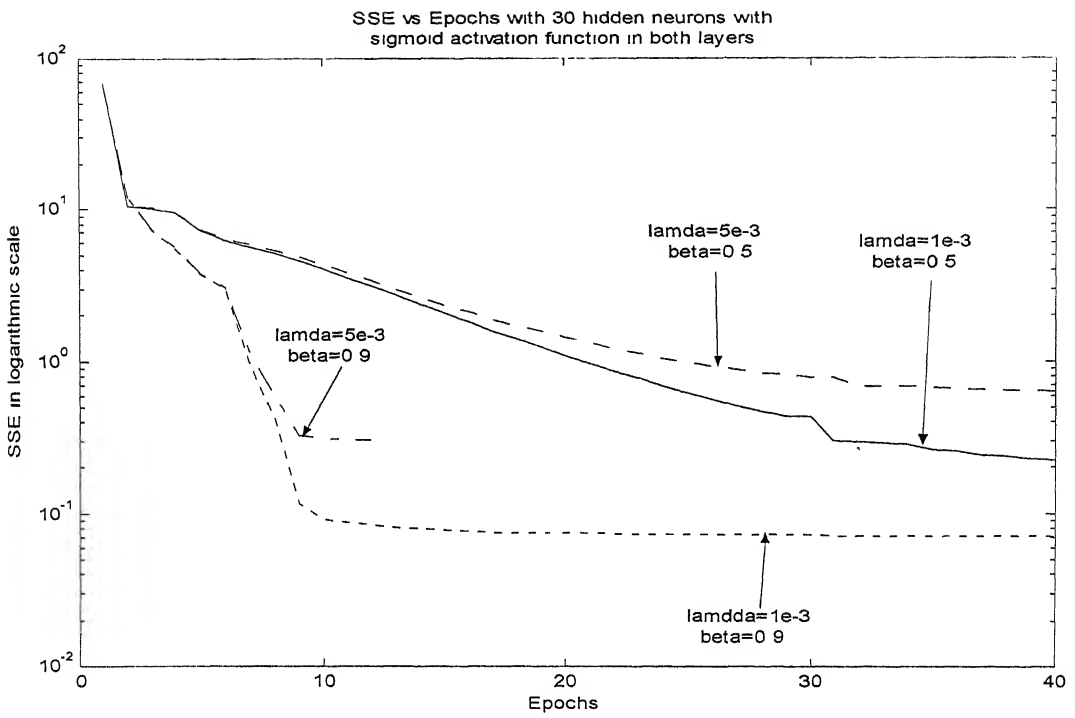


Fig. 4.6-Comparison of SSE curves with different values of lamda

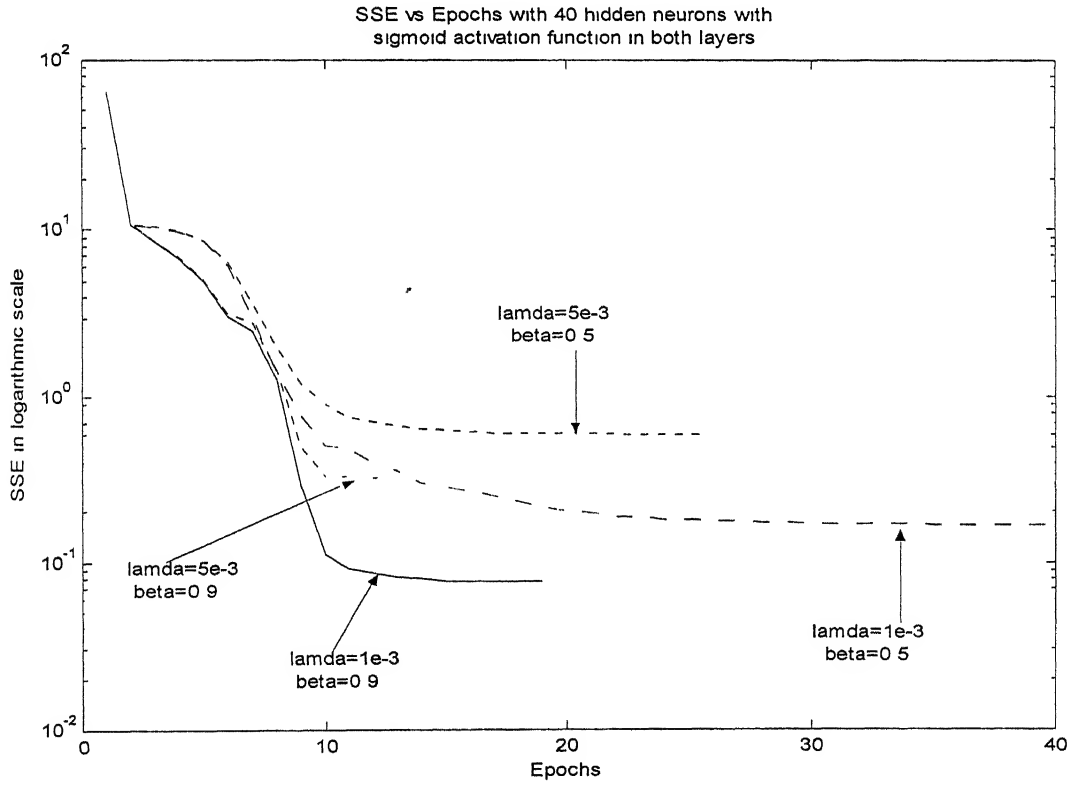


Fig. 4.7-Comparison of SSE curves with different values of lamda

For architectures with sigmoid(beta=0.9) in both layers

Table no 4.15 lamda=1e-3

w1		w2
1.2747	0.4502	0.3931
-1.9151	2.0889	1.8084
-1.3048	-0.3132	0.3733
0.1450	0.1453	-0.4257
1.1080	-0.4750	-0.1792
1.4960	-0.7265	-0 8325
1.5355	-1.7249	-1.5370
-0.8435	1.2498	-0.9595
1.2434	1.6948	1.1510
-0.6502	1.4844	-0.7396
1.8939	1.2635	1.1334
-0.7369	1.3166	-0.7965
-1.5486	0.4594	-1.3910
-0.0853	-0.1897	-0.0831
-1.7425	-0.4731	1.1777
0.5884	-1.4542	-0.5245
-1.4888	-1.5357	-2.5286
1.0229	-0.0907	0.1112
-0.8541	1.2342	-1.1224
-0.1630	-0.7533	0.2842
1.4945	-1.5125	2.2384
-1.0761	-1.7371	1.5202
0.9034	-0.4325	-0.1091
-1.7409	-1.3663	-0.9130
0.4689	1.6877	0.4018
-0.4664	-1.4200	1.0000
1.4605	1.4608	-2.4665
-1.8226	1.8922	2.0982
-1.4654	-0.9199	-0.0218
0.4094	-1.1045	-0.1423

Table no.4.16 lamda=5e-3

w1		w2
0.7672	0.6780	0.4880
-1.6190	1.8760	1.3556
-1.1650	-0.2129	0.2263
0.0416	0.0598	-0.3923
1.1012	-0.3142	-0.3685
1.1814	-0.7809	-0.7672
1.1757	-1.1085	-1.0653
-0.8248	0.9519	-0.8987
1.2355	1.2624	1.2718
-0.6644	1.2788	-0.4898
1.2754	1.2561	1.2066
-0.7782	1.0361	-0.6896
-0.8929	0.8584	-1.3315
-0.0460	-0.1924	-0.0742
-1.3509	-0.5367	0.7440
0.6679	-0.9169	-0.4311
-1.7938	-1.8383	-2.1027
1.0120	-0.0913	-0.1251
-0.8258	0.9218	-1.0816
-0.3428	-0.6073	0.3599
1.9286	-1.9436	1.8918
-0.9502	-1.1677	1.1548
0.7687	-0.4153	-0.3868
-1.6104	-1.3354	-0.4698
0.5246	1.6211	0.0014
-0.6712	-0.9350	0.6320
1.7759	1.7504	-2.0071
-1.8608	1.7303	1.6472
-1.4918	-0.8789	-0.0576
0.3685	-0.9179	-0.1972

For architectures with sigmoid(beta=0.5) in both layers

Table no. 4.17 lamda=5e-3

w1		w2
0.5436	0.6566	0.7027
-2.7828	2.9854	2.6020
-1.2106	0.1704	-0.1381
0.1020	-0.0671	-0.2484
1.0793	-0.2804	-0.7971
1.0587	-0.6897	-1.4140
1.1254	-0.8373	-1.6711
-1.2380	1.3344	-1.7584
1.1994	1.3857	2.0227
-0.9619	1.5458	-1.4244
1.2721	1.4563	2.1244
-1.2246	1.3877	-1.6877
-1.3601	1.3118	-2.0065
-0.0127	-0.1821	-0.1001
-0.9729	-0.3499	0.7164
0.7819	-1.1089	-1.5875
-2.8752	-3.1178	-2.3644
1.5103	-0.0850	-0.0423
-1.0989	1.2450	-1.5929
-0.1385	-0.4833	-0.0113
2.4553	-1.5843	1.9252
-0.8948	-0.6617	1.1483
0.8209	-0.5803	-1.1199
-1.3492	-1.1096	1.4267
1.1442	1.4863	0.5117
-0.2327	-0.4485	0.1375
3.2171	3.0110	-2.3537
-3.2147	3.2021	2.6158
-1.3317	-1.0481	1.3983
1.2801	-2.2331	1.3080

Table no.4.18 lamda=1e-3

w1		w2
0.5788	0.5696	-0.1601
-3.4378	3.2988	3.1322
-1.2134	0.1435	-0.1027
0.0506	0.1219	-0.4624
1.0637	-0.1637	0.0447
2.0635	-1.2905	-1.5705
2.4785	-1.6288	-2.5839
-1.6906	1.4613	-1.9131
2.0754	1.6587	2.1956
-1.2879	1.7437	-1.5436
1.3765	2.1985	2.3488
-1.5588	1.5846	-1.8238
-1.9899	1.2098	-2.3105
-0.0806	-0.1040	-0.2267
-1.2142	-0.2255	0.5105
1.4499	-2.2133	-2.2156
-3.0664	-2.9887	-3.2019
1.5134	-0.0058	0.7446
-1.4702	1.5561	-1.6425
-0.1522	-0.4744	0.0350
3.0417	-2.7222	2.8721
-0.9052	-1.3358	1.1106
1.0418	-0.6062	-0.1378
-1.7181	-1.6063	1.9703
1.0773	1.8125	0.4969
-0.1820	-0.4821	0.0974
2.9723	2.9382	-3.3208
-3.3319	3.2773	2.8226
-2.1360	-1.0108	2.0349
1.6432	-1.9673	1.2937

For architectures with sigmoid(beta=0.5) in both layers

Table 4.19 lamda=1e-3

w1		w2
2.8106	2.7065	-1.3551
-3.0296	-3.0033	-2.8832
0.2730	0.5419	0.2985
0.0594	-0.2618	0.0236
1.9415	1.0813	1.4924
0.6731	0.0365	-0.3801
-0.2858	0.6363	-0.2913
-1.5129	0.1422	-0.2931
1.6595	-1.4186	-2.0016
-1.5843	-1.7964	2.3900
0.2081	-0.7827	-0.1316
1.2124	1.4956	1.3659
3.0771	-2.9764	2.8786
0.5918	0.0682	-0.3175
-1.7171	-1.4772	-0.4836
-0.5698	0.9133	-0.5329
1.4813	-0.5231	-0.9594
2.9911	3.0262	-2.1117
-0.1467	0.8835	-0.0663
1.2347	0.8857	0.7283
-1.2143	0.4799	-0.2411
-0.1012	1.2912	0.0877
1.5006	1.8872	2.3111
-1.7241	0.5845	-0.7939
-2.7327	2.8032	1.7644
-1.5412	1.8768	-2.1061
-1.6163	-0.9804	1.6749
0.6520	-0.6449	-0.6314
-1.4635	-0.9738	1.5516
-1.1527	1.0931	-0.8012
-2.8227	2.9268	1.9261
1.6150	-1.4742	-2.0039
0.0613	0.9026	0.2175
1.1733	0.5604	0.3695
0.0225	-0.2569	0.0599
-0.4791	0.8221	-0.4522
0.8519	0.3819	0.0710
0.2732	-0.1127	-0.3385
-1.5269	1.8463	-2.0351
0.3742	0.3274	0.0942

Table no. 4.20 lamda=5e-5

w1		w2
2.7479	2.6417	-1.3034
-2.7445	-2.7691	-2.6138
0.3416	0.5554	0.4230
0.0514	-0.2608	0.0273
1.6164	1.1178	1.1465
0.6787	-0.0467	-0.3814
-0.3590	0.6446	-0.4658
-1.4525	0.1259	-0.0672
1.2536	-1.0052	-1.7610
-1.3382	-1.3044	2.1674
0.2391	-0.7670	-0.1756
1.0173	1.1617	1.2999
2.9463	-2.8931	2.6857
0.5967	-0.0033	-0.3076
-1.8263	-1.4655	-0.6599
-0.5941	0.8348	-0.7596
1.2170	-0.4750	-0.9543
2.9415	2.8955	-1.8756
-0.1743	0.8782	-0.1665
1.0441	0.8215	0.8026
-1.1164	0.4786	-0.2686
-0.1055	1.3015	0.0591
1.2663	1.4997	1.9963
-1.3903	0.5709	-0.4664
-2.4504	2.5756	1.6218
-1.2777	1.4625	-1.7554
-1.2085	-0.8293	1.6525
0.6698	-0.6128	-0.8043
-1.1127	-0.8394	1.6015
-0.9336	0.9358	-0.8921
-2.6702	2.5894	1.7807
1.2377	-1.0083	-1.7603
0.1199	0.8985	0.2470
1.0758	0.5731	0.4580
0.0163	-0.2579	0.0613
-0.5284	0.7787	-0.6745
0.8385	0.4002	0.2128
0.2957	-0.1570	-0.3700
-1.2583	1.4332	-1.7009
0.3927	0.3493	0.2095

For architectures with sigmoid(beta=0.9) in both layers

Table no. 4.21 lamda=1e-3

w1		w2
1.8040	1.6866	-0.8900
-1.5345	-1.4774	-1.0759
0.4942	1.1053	0.4286
-0.0588	-0.3132	-0.0383
1.8692	1.7552	-0.9452
0.6514	0.2196	-0.2610
-0.1633	0.5167	-0.0689
-1.4930	-0.5836	0.0105
1.5570	-0.4101	-0.9072
-1.1144	-1.7169	1.3652
0.4254	-0.9731	-0.1442
1.1542	1.1800	1.1037
1.5413	-1.4454	2.7321
0.5811	0.1886	-0.2154
-1.7705	-1.5451	-2.1558
-0.5361	1.0442	-0.4160
1.4283	-0.3524	-0.5823
1.9311	1.7613	-1.8133
-0.2756	0.7619	0.1339
1.8179	0.4869	1.0229
-1.2482	0.2608	-0.3008
-0.1261	1.1866	-0.1317
1.1951	1.6246	1.4152
-1.2632	-0.0369	0.0647
-1.8624	1.7942	1.8076
-1.1819	1.4959	-1.3551
-1.7135	-0.6434	1.0698
0.5554	-1.1509	-0.4405
-1.4144	-0.8645	0.9426
-1.4528	0.4745	-0.6581
-1.8838	1.8090	1.2618
1.3885	-1.6649	-1.5362
-0.0415	0.8800	0.1200
1.3706	0.4019	0.3851
-0.0710	-0.3017	-0.0350
-0.4360	0.8360	-0.2324
1.0539	0.3578	0.1355
0.1139	-0.0533	-0.2776
-1.3733	1.1832	-1.1919
0.3613	0.3148	-0.0292

Table no. 4.22 lamda=5e-3

w1		w2
1.5838	1.5104	-0.7196
-1.1657	-1.3816	-0.7159
0.4737	0.8511	0.3826
-0.0390	-0.3105	-0.0584
1.6346	1.5680	-0.7634
0.6484	0.1043	-0.1634
-0.2340	0.5114	-0.1906
-1.4935	-0.5933	-0.1715
1.2234	-0.7046	-0.7563
-0.9968	-1.3626	1.1387
0.4431	-0.8586	-0.2620
1.0559	1.0815	1.0455
1.7869	-1.7691	2.3470
0.5689	0.0857	-0.1373
-1.8030	-1.7635	-1.9173
-0.5693	0.7932	-0.4162
1.2338	-0.3274	-0.3964
1.7677	1.7316	-1.6372
-0.2471	0.7826	0.0196
1.3537	0.6402	0.6663
-1.0876	0.2143	-0.1486
-0.1036	1.1812	-0.0273
1.0642	1.3454	1.2132
-1.2790	0.0441	0.1438
-1.8371	1.8924	1.5907
-1.1348	1.2566	-1.2189
-1.2628	-0.8624	0.9204
0.6392	-0.9039	-0.4989
-1.1414	-0.9195	0.8959
-1.0280	0.5458	-0.4303
-1.5752	1.4504	1.0238
1.0901	-1.2444	-1.2115
0.0079	0.9042	0.1755
1.1842	0.3494	0.2413
-0.0509	-0.2999	-0.0510
-0.4621	0.6960	-0.3321
0.9663	0.3530	0.1589
0.1459	-0.1172	-0.3285
-1.1399	1.1650	-1.0912
0.3769	0.3355	0.1227

Discussion-

From the above weight matrix table, it is evident that this method does not produce any significant reduction in weight values. In the weight table, we find some of the weights are reduced to minimum i.e. less than 0.001 and some of the weights are saturated around 1.5 to 2.5. The error surface is very much sensitive to the regularization parameter (lamda). The increase in the value of lamda saturates the network and the network does not converge to desired minimum value

- **Weight elimination method**

This pruning process is tried with different regularization parameter (lamda) with both 30 and 40 hidden neuron architecture. The error surface with different regularization parameter is given in fig. 4.8 and fig. 4.9 for 30 and 40 hidden neurons respectively. The weight values after the training process are listed in table 4.23,4.24,4.25,4.26,4.27,4.28 give an illustration of the pruning algorithm. In tables,w1 indicates the input to hidden layer weights,w2 indicates the hidden to output layer weights.

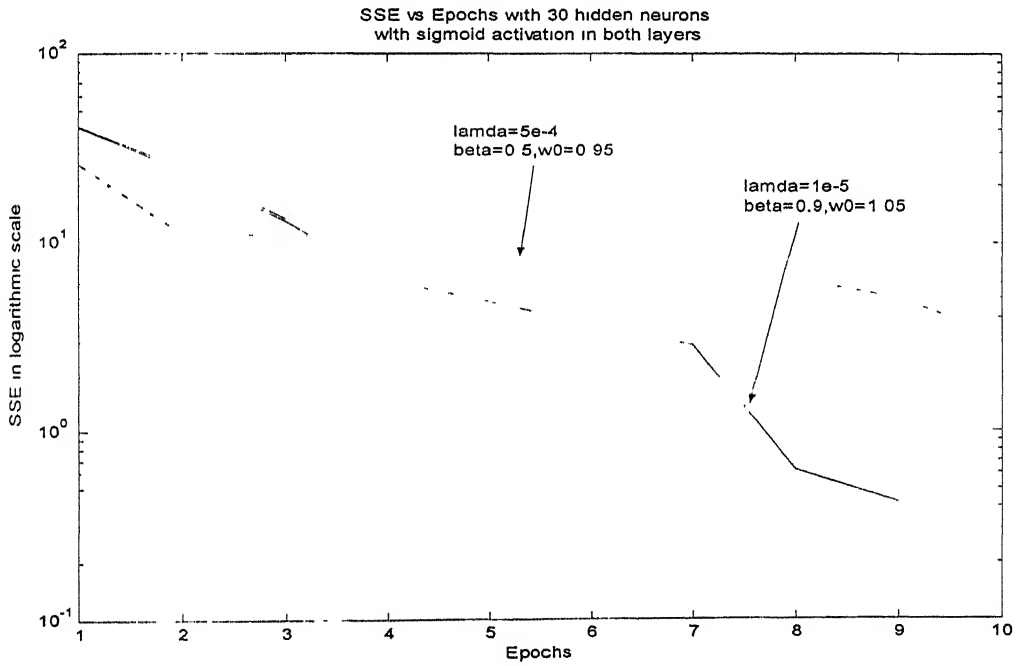


Fig. 4.8-Comparison of SSE curves with different values of lamda,w0

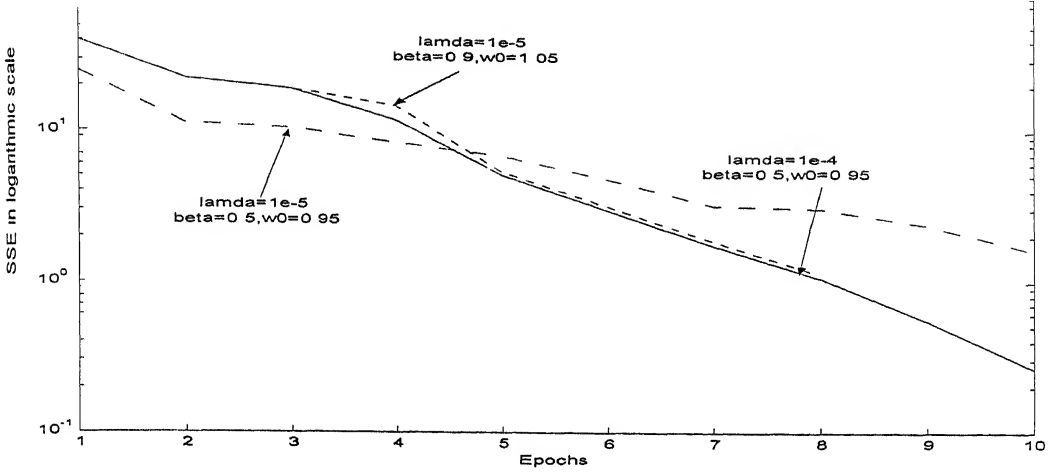


Fig. 4.9-Comparison of SSE curves with different values of lamda,w0
For architectures with sigmoid(beta=0.9) in both layers

Table no. 4.23 lamda=1e-5,w0=1.05

w1		w2
0.4962	0.5287	0.5522
-1.9547	2.6675	1.6173
-1.1390	-0.2035	0.5130
0.0695	-0.0252	-0.3622
0.9661	-0.0129	-0.1757
0.8736	-0.6024	-0.8359
0.9075	-0.7260	-1.0467
-0.8873	0.8677	-0.9428
0.8160	1.0291	1.2668
-0.7971	1.0733	-0.4300
0.8728	1.0694	1.3194
-0.8798	0.9137	-0.7055
-0.9387	0.9108	-1.3583
0.1198	-0.1512	-0.1525
-1.0982	-0.5618	0.9555
0.5114	-0.7492	-0.5825
-1.9505	-2.4618	-1.9553
1.2583	0.1035	0.0182
-0.8895	0.8763	-1.0947
-0.3027	-0.4530	0.3538
2.1447	-1.9483	1.9433
-0.8334	-0.6671	0.9734
0.7623	-0.6556	-0.8339
-1.6819	-1.1457	-0.5299
0.7205	1.7487	-0.2604
-0.4897	-0.5516	0.5462
2.5389	2.1250	-2.0952
-2.5103	1.8477	1.4706
-1.6471	-0.8867	-0.3305
0.5338	-1.6680	0.7280

Table no.4.24 lamda=5e-4,w0=0.95

w1		w2
0.3974	0.6703	0.5484
-3.4828	4.8101	3.5281
-1.1555	0.6799	-0.2458
0.0880	-0.0400	-0.1892
0.9370	1.1785	1.1231
0.9095	-0.1680	-0.8734
1.0509	-0.4674	-1.4113
-1.4417	1.5851	-2.3407
0.9313	1.5021	2.0326
-1.3472	1.5630	-1.9197
1.2445	1.8243	2.9261
-1.4480	1.5978	-2.3199
-1.4622	1.5955	-2.4775
-0.0522	-0.1642	-0.0276
-1.0152	0.1938	0.4566
0.8668	-0.8899	-1.6348
-3.2663	-4.3604	-4.0657
1.3186	1.1595	0.8907
-1.3602	1.5155	-2.0789
-0.2791	-0.4756	0.2096
3.2729	-1.5807	2.5796
-1.0931	-0.6230	1.5902
0.8713	-0.4887	-1.2804
-1.4282	-0.8969	1.5666
1.4756	2.9654	-0.6040
-0.3568	-0.4534	0.3301
4.7666	3.6194	-3.7601
-4.7534	3.7608	3.7352
-1.4482	-0.8982	2.1820
1.4863	-3.0799	2.5110

For architectures with sigmoid(beta=0.9) in both layers

Table no. 4.25- lamda=1e-3,w0=0.95

w1		w2
0.4066	0.2404	0.1274
-1.4779	1.9499	0.8128
-1.0920	0.0971	0.0020
-0.0295	0.0390	-0.5523
0.9925	-0.0864	0.2022
0.7333	-0.2105	-0.0662
0.7303	-0.2709	-0.1151
-0.7698	0.7279	-0.4952
0.4567	0.7170	0.4474
-0.7597	1.1310	-0.0149
0.5176	0.6459	0.4373
-0.8696	0.9440	-0.2273
-0.7238	0.5442	-0.9569
-0.0120	-0.0956	-0.2907
-0.9904	-0.0450	0.1635
0.1859	-0.6116	-0.0511
-1.2343	-1.5725	-0.0306
1.2614	-0.0025	0.3915
-0.7020	0.4866	-0.7340
-0.0507	-0.4407	0.0734
1.1750	-0.6983	0.3199
-0.4135	-0.2461	0.2202
0.4150	-0.1800	-0.2632
-1.4884	-0.9230	0.0502
0.5380	1.2635	-0.1090
-0.0848	-0.4994	0.1430
1.7369	1.1804	-0.2479
-1.7732	1.5697	0.7598
-1.4488	-0.7775	0.0802
0.2974	-1.0841	0.3178

For architectures with sigmoid(beta=0.5) in both layers

Table no.4.26 lamda=1e-5,w0=0.95

w1		w2
3.7968	3.4573	-2.6727
-3.5322	-3.8412	-4.1004
0.3055	0.5225	0.5219
0.2086	-0.2567	-0.2632
2.3731	2.0476	-0.0931
0.7088	-0.1253	-0.4793
-0.5822	0.7715	-0.7187
-1.5471	0.0787	0.4378
1.2586	-0.8716	-2.4869
-1.7646	-1.6547	3.6532
-1.3177	-1.1984	2.0970
1.1806	1.4963	2.0864
3.5468	-3.8128	4.1904
0.5842	-0.0694	-0.3669
-3.0558	-1.4007	-1.6333
-0.9484	1.0897	-1.2983
1.5466	-0.4572	-0.6492
3.2862	3.7331	-2.9268
0.2104	0.9941	0.4624
1.1122	1.2896	1.6062
-1.0383	0.8235	-0.6684
0.2183	1.5890	0.4420
1.5296	1.5865	3.0930
-1.5834	0.3625	0.0901
-3.3244	4.1433	3.1740
-1.5630	1.6648	-2.7082
-0.9761	-0.4657	1.1666
0.8433	-0.6728	-1.1748
-0.8196	-0.5230	1.1391
-1.2107	1.3655	-1.7903
-3.7082	2.6613	2.6136
1.6324	-1.0301	-3.0600
0.5239	1.0612	0.9227
1.1459	1.2485	1.5325
0.1495	-0.2212	-0.1791
-0.8557	1.0082	-1.1421
0.9041	0.8006	0.8296
0.4654	-0.2971	-0.6456
-1.6245	1.7216	-2.8922
0.3363	0.3203	0.2617

Table no.4.27 lamda=1e-4,w0=0.95

w1		w2
1.6795	1.1270	-0.3594
-0.8318	-1.1581	-0.7018
0.3573	0.5115	0.4553
0.2000	-0.2810	-0.0473
1.9155	1.8323	-1 4788
0.7739	-0.2147	-0.2592
-0.6391	0.7065	-0.5286
-1.7230	-0.4548	-0.7325
0.8732	-0.5603	-0.4338
-1.2159	-1.1258	1.3986
0.4048	-0.8759	-0.4305
0.9712	1.0126	1.1109
2.0804	-2.0791	2.3261
0.6844	-0.2277	-0.2322
-1.9760	-2.0371	-2.3856
-0.7749	0.7877	-0.7517
1.1043	-0.2214	-0.3288
1.9771	2.0748	-1.7276
-0.0401	0.8615	0.0548
1.0039	0.7670	0.7066
-1.0669	0.2522	-0.1934
-0.6506	1.6782	0.2681
1.0391	1.0409	1.0428
-1.3847	-0.0340	0.0601
-2.0256	2.2851	2.1614
-0.9274	0.8886	-1.0513
-1.0433	-0.9254	1.1404
0.7650	-0.8932	-0.7597
-0.9567	-0.8559	1.0649
-0.8774	0.7708	-0.7385
-2.1484	1.0262	1.1449
1.0863	-1.1591	-1.3221
0.2541	0.9877	0.3747
1.0285	0.4091	0.2739
0.1655	-0.2378	-0.0202
-0.7494	0.7732	-0.6985
0.8282	0.3368	0.2438
0.3611	-0.2849	-0.3202
-0.9979	0.8640	-1.0497
0.3480	0.2186	0.2561

For architectures with sigmoid(beta=0.9) in both layers

Table no. 4.28-lamda=1e-5,w0=1.05

w1		w2
1.6097	1.0648	-0.2882
-0.6507	-1.2503	-0.7214
0.3768	0.5082	0.4066
0.1909	-0.2601	-0.1268
2.0962	1.5621	-1.2599
0.6997	-0.1870	-0.2396
-0.5386	0.6019	-0.5609
-1.6938	-0.3716	-0.7756
0.8054	-0.5382	-0.3233
-0.9026	-0.8857	1.2759
0.3647	-0.7352	-0.3459
0.8207	0.8003	0.9385
2.1240	-2.3558	1.6803
0.6168	-0.2022	-0.2410
-1.7638	-1.8225	-1.9985
-0.6514	0.6742	-0.7450
1.0793	-0.1754	-0.1747
1.9801	2.2343	-1.4233
-0.0431	0.8614	0.0107
0.8824	0.6055	0.5756
-1.0346	0.2386	-0.1012
-0.5311	1.6246	0.4276
0.8433	0.7549	0.9065
-1.4055	0.0481	0.1932
-2.1288	2.3153	1.5532
-0.7753	0.7763	-1.0043
-0.8417	-0.7401	1.0578
0.6333	-0.6918	-0.6473
-0.7746	-0.7127	1.0016
-0.7326	0.6789	-0.7154
-2.1347	1.1633	0.9558
0.9087	-0.9000	-1.0487
0.2081	0.9408	0.2994
0.9954	0.3244	0.2209
0.1586	-0.2219	-0.1095
-0.6278	0.6583	-0.6994
0.8235	0.2864	0.2226
0.3099	-0.2336	-0.3915
-0.7989	0.7458	-1.0015
0.3768	0.2297	0.1933

Discussion-

From the above weight matrix table, it is evident that this method does not produce any significant reduction in weight values. In the weight table, we find some of the weights are reduced to minimum i.e. less than 0.001 and some of the weights are saturated around 1.5 to 2.5. The error surface is very much sensitive to the regularization parameter (lamda) and the free parameter w0. The increase in the value of lamda saturates the network and the network does not converge to desired minimum value. In some of the simulations, it is observed that when the value of w0 is set to 1.1 the network saturates very quickly without any significant reduction in error. The point to be noted here that this method gives better results than the other two penalty functions and it saturates much faster than those methods.

Sensitivity Analysis Method-

In this method of pruning, the network is trained normally done. During the training process, in each epoch the weight sensitivities are calculated according to the formula given in chapter-3. After training procedure, the low sensitivity weights are deleted. In this method, the threshold sensitivity is determined in such a manner that after deletion of weight the increase in SSE is within certain range. The process is tried with both 30 and 40 hidden neuron architecture using sigmoid activation function in both output and hidden layer. The pruned weights are given in the table 4.29,4.30,4.31,4.32.

For architectures having 30 hidden neuron &
sigmoid activation function in both layers

Threshold input-hidden layer weight sensitivity, Sih≤1.0

Threshold hidden-output layer weight sensitivity, Sho≤3.0

Table no. 4.29 – beta=0.9
SSE after pruning=0.2571

w1		w2
0.5516	0.5111	0.1468
-2.6187	2.3515	2.6574
-1.9025	-0.7241	1.4856
0	0	-0.3379
0	0	0
1.6373	-0.7795	-1.0204
2.5756	-2.0210	-2.5990
-2.6403	1.1654	-1.8788
1.8021	1.4379	1.6554
-0.3323	2.0042	-1.2627
2.1915	2.0596	2.2040
-0.9219	1.2638	-1.0269
-1.8013	1.8072	-2.2956
0	0	-0.0172
-2.5486	-1.0049	2.2400
0.8096	-1.9494	-0.7450
-2.5245	-2.2987	-3.4774
1.3866	0	0.3847
-1.4095	1.2682	-1.5571
0	-0.5897	0.4586
1.6640	-1.6585	3.6590
-1.8372	-1.7375	2.3831
1.5134	-0.9163	-0.9413
-3.0610	-2.4432	-2.6761
0.4640	2.3779	0.2770
-0.8039	-2.1943	1.4869
1.7753	1.5725	-2.7561
-2.8069	2.7287	2.8900
-1.5144	-1.1522	-0.1024
0	-2.8853	0.4126

Table no. 4.30 –beta=0.5

SSE after pruning=0.8372

w1		w2
0	0.6724	0.4055
-4.6442	5.2672	5.3065
-1.2447	0	0.5792
0	0	-0.4234
1.2517	0	0.5648
2.4720	-1.2216	-2.2247
3.7551	-1.1303	-3.8345
-1.9046	2.2078	-3.6444
3.0226	2.7407	5.3583
-1.3378	2.6435	-2.7068
0	2.9832	3.8877
-1.7567	2.2438	-3.4596
-2.1361	2.2453	-4.0666
0	0	-0.2895
-1.3497	0	1.0050
1.3486	-2.0349	-1.7661
-3.4889	-3.6538	-6.4223
2.1366	0	0 5736
-1.5768	2.2548	-3.1020
0	0	0.2735
4.4040	0	3.3227
-1.9670	-0.8934	2.2039
2.1657	-1.4849	-2.1534
-1.7778	0	0.9136
2.3927	3.9117	-2.0548
0	-0.4773	0.3359
3.8869	3.0484	-3.8976
-4.7953	4.8981	4.4284
-2.6538	-0.9286	2.6145
3.4843	-4.1095	4.7922

For architectures having 40 hidden neuron &

sigmoid activation function in both layers

Threshold input-hidden layer weight sensitivity, Sih≤0.8

Threshold hidden-output layer weight sensitivity, Sho≤3.0

Table no. 4.31–beta=0.5
SSE after pruning=1.2571

w1		w2
4.4239	4.7093	-4.1866
-3.3152	-3.5234	-5.2410
0.2446	0.4159	0.5069
0	0	-0.0818
2.6375	1.8874	0.7562
0	0	-0.0793
0	0	-0.5545
-1.7188	0	-0.0377
2.7811	-1.1290	-2.9211
-3.7889	-3.2055	4.9770
-2.6881	-1.2430	1.7616
1.1611	2.9898	3.1299
3.7253	-3.5697	6.6990
0.5723	0	-0.0313
-3.2882	-1.0418	-1.0287
0	1.7183	-1.3762
0	-0.7108	-0.8835
4.2284	5.0505	-4.9106
0	0	0.3554
1.1827	2.3310	2.1232
-1.3888	0	-0.8490
0	1.7615	0.5163
2.6205	2.8850	4.5438
-1.6227	0.3180	-0.4960
0	4.0762	3.8370
-1.3947	2.7699	-3.4672
-1.0398	0	0.2236
1.0605	-0.8634	-0.6367
-0.6523	0	0
-0.8275	2.6512	-2.4917
-4.1395	3.1120	3.7699
2.8236	-2.6068	-4.1353
0	1.2774	0.8045
1.3253	2.0462	1.9316
0	0	-0.0396
0	1.3899	-1.0981
1.2285	0	0.8660
0	0	-0.2659
-2.7624	2.0468	-3.9921
0	0	0.3363

Table no. 4.32 –beta=0.9
SSE after pruning=0.8372

w1		w2
1.4839	1.5607	-0.5764
-1.2017	-0.8642	-0.9779
0.2098	0	0.1599
0	0	-0.0572
2.5135	2.5317	-2.5677
0.8136	0	-0.1369
0	1.1067	-0.4603
-2.0634	-0.6632	-0.5299
1.5366	-0.3210	-0.7507
-2.6797	-2.0844	2.7253
0	-1.1101	-0.1774
0.9409	2.4933	2.0819
1.6254	-1.5952	3.7181
0.6780	0	-0.0886
-1.9043	-1.7358	-3.2203
-0.3580	1.3938	-0.8539
1.7004	0	-0.9284
2.5248	2.4309	-2.6476
0	0.8795	0.1704
1.6733	0.5614	1.4099
-2.2863	0	-1.4245
0	2.5601	-0.9589
1.3527	2.4933	2.6322
-1.4771	0	-0.4001
-1.4862	1.8215	3.7032
-0.3032	2.2584	-1.3452
-1.6764	-1.0984	1.2551
1.0766	-2.2354	-0.9794
-3.1360	-1.1133	1.7454
-1.6550	0.3387	-1.2173
-2.5285	1.3907	1.9071
2.3943	-2.4749	-2.5496
0	1.7969	0.9765
1.5217	0	0.9260
0	0	-0.0707
-0.3762	1.3460	-0.7692
1.0791	0	0.3185
0	-0.1356	-0.2801
-2.1157	2.0896	-2.6645
0	0	0

RESULTS OF ARCHITECTURES OF 30 HIDDEN NEURONS HAVING SIGMOID ACTIVATION IN BOTH LAYERS WITHOUT PRUNING

Table no. 4.33–beta=0.9
Final SSE=0.001

w1		w2
0.5516	0.5111	0.1468
-2.6187	2.3515	2.6574
-1.9025	-0.7241	1.4856
0.0524	0.1335	-0.3379
1.0250	-0.0329	0.1215
1.6373	-0.7795	-1.0204
2.5756	-2.0210	-2.5990
-2.6403	1.1654	-1.8788
1.8021	1.4379	1.6554
-0.3323	2.0042	-1.2627
2.1915	2.0596	2.2040
-0.9219	1.2638	-1.0269
-1.8013	1.8072	-2.2956
0.0748	-0.0857	-0.0172
-2.5486	-1.0049	2.2400
0.8096	-1.9494	-0.7450
-2.5245	-2.2987	-3.4774
1.3866	0.1430	0.3847
-1.4095	1.2682	-1.5571
-0.0058	-0.5897	0.4586
1.6640	-1.6585	3.6590
-1.8372	-1.7375	2.3831
1.5134	-0.9163	-0.9413
-3.0610	-2.4432	-2.6761
0.4640	2.3779	0.2770
-0.8039	-2.1943	1.4869
1.7753	1.5725	-2.7561
-2.8069	2.7287	2.8900
-1.5144	-1.1522	-0.1024
0.2648	-2.8853	0.4126

Table no. 4.34–beta=0.5
Final SSE=0.0707

w1		w2
0.3669	0.6724	0.4055
-4.6442	5.2672	5.3065
-1.2447	-0.0839	0.5792
0.0441	-0.0118	-0.4234
1.2517	0.5492	0.5648
2.4720	-1.2216	-2.2247
3.7551	-1.1303	-3.8345
-1.9046	2.2078	-3.6444
3.0226	2.7407	5.3583
-1.3378	2.6435	-2.7068
0.9799	2.9832	3.8877
-1.7567	2.2438	-3.4596
-2.1361	2.2453	-4.0666
-0.0241	-0.1181	-0.2895
-1.3497	-0.3557	1.0050
1.3486	-2.0349	-1.7661
-3.4889	-3.6538	-6.4223
2.1366	0.1180	0.5736
-1.5768	2.2548	-3.1020
-0.4661	-0.4553	0.2735
4.4040	-1.8197	3.3227
-1.9670	-0.8934	2.2039
2.1657	-1.4849	-2.1534
-1.7778	-1.1171	0.9136
2.3927	3.9117	-2.0548
-0.5240	-0.4773	0.3359
3.8869	3.0484	-3.8976
-4.7953	4.8981	4.4284
-2.6538	-0.9286	2.6145
3.4843	-4.1095	4.7922

RESULTS OF ARCHITECTURES OF 30 HIDDEN NEURONS HAVING SIGMOID ACTIVATION IN BOTH LAYERS WITHOUT PRUNING

Table no. 4.35-beta=0.5
Final SSE=0.0547

w1		w2
4.4239	4.7093	-4.1866
-3.3152	-3.5234	-5.2410
0.2446	0.4159	0.5069
0.1388	-0.2563	-0.0818
2.6375	1.8874	0.7562
0.7065	-0.0113	-0.0793
-0.6075	0.7862	-0.5545
-1.7188	0.0717	-0.0377
2.7811	-1.1290	-2.9211
-3.7889	-3.2055	4.9770
-2.6881	-1.2430	1.7616
1.1611	2.9898	3.1299
3.7253	-3.5697	6.6990
0.5723	0.0125	-0.0313
-3.2882	-1.0418	-1.0287
-0.8548	1.7183	-1.3762
1.9404	-0.7108	-0.8835
4.2284	5.0505	-4.9106
0.1397	0.9841	0.3554
1.1827	2.3310	2.1232
-1.3888	0.6401	-0.8490
0.4097	1.7615	0.5163
2.6205	2.8850	4.5438
-1.6227	0.3180	-0.4960
-2.9328	4.0762	3.8370
-1.3947	2.7699	-3.4672
-1.0398	-0.2066	0.2236
1.0605	-0.8634	-0.6367
-0.6523	-0.2017	0.2600
-0.8275	2.6512	-2.4917
-4.1395	3.1120	3 7699
2.8236	-2.6068	-4.1353
0.5936	1.2774	0.8045
1.3253	2.0462	1.9316
0.1109	-0.2236	-0.0396
-0.8810	1.3899	-1.0981
1.2285	0.7721	0.8660
0.3281	-0.2221	-0.2659
-2.7624	2.0468	-3.9921
0.3407	0.2369	0.3363

Table no. 4.36–beta=0.9
Final SSE=0.0099

w1		w2
1.4839	1.5607	-0.5764
-1.2017	-0.8642	-0.9779
0.2098	0.5219	0.1599
0.1623	-0.2711	-0.0572
2.5135	2.5317	-2.5677
0.8136	-0.1063	-0.1369
-0.4923	1.1067	-0.4603
-2.0634	-0.6632	-0.5299
1.5366	-0.3210	-0.7507
-2.6797	-2.0844	2.7253
0.3940	-1.1101	-0.1774
0.9409	2.4933	2.0819
1.6254	-1.5952	3.7181
0.6780	-0.0962	-0.0886
-1.9043	-1.7358	-3.2203
-0.3580	1.3938	-0.8539
1.7004	-0.1735	-0.9284
2.5248	2.4309	-2.6476
-0.0448	0.8795	0.1704
1.6733	0.5614	1.4099
-2.2863	0.5664	-1.4245
-0.8815	2.5601	-0.9589
1.3527	2.4933	2.6322
-1.4771	0.0833	-0.4001
-1.4862	1.8215	3.7032
-0.3032	2.2584	-1.3452
-1.6764	-1.0984	1.2551
1.0766	-2.2354	-0.9794
-3.1360	-1.1133	1.7454
-1.6550	0.3387	-1.2173
-2.5285	1.3907	1.9071
2.3943	-2.4749	-2.5496
0.5316	1.7969	0.9765
1.5217	0.5547	0.9260
0.1367	-0.2323	-0.0707
-0.3762	1.3460	-0.7692
1.0791	0.3469	0.3185
0.1977	-0.1356	-0.2801
-2.1157	2.0896	-2.6645
0.3103	0.1128	0.0211

CENTRAL LIBRARY
I. I. T., KANPUR

sin(x)sin(y) OUTPUT TABLE FOR DIFFERENT PATTERNS

Table 4.37 FOR ARCHITECTURE HAVING 30 HIDDEN NEURON

ıut patterns in radian)		output pattern	case 1	case 2	case 3	case 4	case 5	case 6	case 7	case 8	case 9	case 10	case 11	case 12
00	0.1000	0.0100	-0.0052	-0 0259	-0.0064	-0 0095	-0.0664	0.0079	-0.0576	-0.0140	-0.0935	-0.0584	-0.0095	-0.0064
00	1.1000	0.0890	0.0907	0 1020	0.0876	0.0933	0.1646	0.0881	0.1411	0.0945	0.1597	0 2149	0 0933	0.0876
00	2.1000	0.0862	0.0830	0.0632	0.0844	0.0860	0.0386	0.0831	0.0727	0 0770	-0.0191	0.0882	0.0860	0.0844
00	3.1000	0.0042	-0.0081	0.0265	-0 0105	-0.0264	-0.0302	-0.0176	0 0051	-0.0153	-0 0647	-0.0149	-0 0264	-0.0105
00	4.1000	-0.0817	-0.1195	-0 1375	-0.1179	-0.0893	-0.0628	-0 1122	-0.1163	-0 1088	-0.0577	-0.0749	-0.0893	-0 1179
00	5.1000	-0.0924	-0.1298	-0 1219	-0.1259	-0.1695	-0.2225	-0 1368	-0.2444	-0.1446	-0.2197	-0.2177	-0 1695	-0.1259
00	6.1000	-0.0182	-0.0380	-0 0416	-0.0397	-0.0176	0.0051	-0.0354	0.0694	-0.0308	-0.0304	-0.0308	-0.0176	-0.0397
00	0.1000	0.0890	0.0875	0.1000	0 0869	0 0923	0.1777	0.0855	0.1413	0.0981	0.1381	0.2014	0.0923	0.0869
00	1.1000	0.7943	0.8933	0 9025	0.9279	0 8891	0.8607	0 9058	0.9221	0.9168	0.8237	0.7931	0.8891	0.9279
00	2.1000	0.7693	0.9081	0.9101	0.8787	0.9068	0.8189	0 9145	0.8219	0.8995	0.7856	0.7429	0 9068	0.8787
00	3.1000	0.0371	0 0277	0 0077	0 0310	0.0259	0 0603	0 0321	0.0524	0.0291	0 0337	0.1886	0.0259	0.0310
00	4.1000	-0.7293	-0.9221	-0 9173	-0 9105	-0.9166	-0.8215	-0.9228	-0.8000	-0.8901	-0.8079	-0.6357	-0.9166	-0.9105
00	5.1000	-0.8251	-0.9228	-0.9497	-0 9584	-0.9170	-0.8998	-0.9313	-1 0174	-0.9827	-0 8872	-0 7888	-0 9170	-0.9584
00	6.1000	-0.1623	-0.2126	-0 2115	-0 2109	-0.2326	-0.3021	-0.2187	-0.3065	-0.2236	-0.3035	-0.2784	-0.2326	-0.2109
00	0 1000	0.0862	0.0936	0 0720	0 0849	0.0837	0.0069	0.0922	0 0433	0 0765	0 0194	0.0605	0 0837	0.0849
00	1.1000	0.7693	0 9046	0.9440	0 8934	0 9121	0.8436	0 8948	0.8832	0.9051	0.8171	0 8361	0 9121	0 8934
00	2.1000	0.7451	0.8719	0.9048	0 8751	0.8412	0 8978	0.8568	0.9079	0.8655	0.8755	0.8632	0.8412	0 8751
00	3.1000	0.0359	0.0363	0.0323	0.0284	0.0362	0.0049	0.0225	-0.0227	0.0235	-0.0441	-0.0427	0 0362	0.0284
00	4.1000	-0.7063	-0.8851	-0.8736	-0.8788	-0.8716	-0.8944	-0.8905	-0.8836	-0.8599	-0.8935	-0 8101	-0.8716	-0.8788
00	5.1000	-0.7992	-0.9331	-0.9317	-0.9499	-0 9488	-0.8614	-0.9286	-0 8876	-0.9791	-0.8461	-0 6831	-0.9488	-0.9499
00	6.1000	-0.1572	-0.2115	-0 2143	-0.2090	-0.2022	-0.1270	-0 2046	-0.2044	-0.2008	-0.1638	-0 1500	-0.2022	-0 2090
00	0.1000	0.0042	-0.0227	-0.0148	-0 0113	-0 0145	0 0412	-0.0287	0 0035	-0 0186	-0 0357	0.0247	-0.0145	-0.0113
00	1.1000	0.0371	0.0269	0.0326	0.0223	0.0135	-0 0213	0.0292	0.0083	0 0216	-0.0243	-0.0769	0.0135	0.0223
00	2.1000	0.0359	0.0282	0.0112	0.0314	0.0496	0.0647	0.0190	0.0551	0.0113	0.0757	0.1796	0.0496	0.0314
00	3.1000	0.0017	-0.0326	-0 0088	-0.0233	-0.0398	0.0114	-0.0076	-0.0070	-0.0132	-0.0700	-0.0386	-0.0398	-0.0233
00	4.1000	-0.0340	-0.0611	-0.0611	-0.0537	-0.0667	-0.1206	-0.0686	-0.1031	-0 0591	-0.1626	-0.1296	-0.0667	-0.0537
00	5.1000	-0.0385	-0.0737	-0.0727	-0.0678	-0.0719	-0.0126	-0.0650	-0.0446	-0.0642	-0.0894	0.0872	-0.0719	-0.0678
00	6.1000	-0.0076	-0.0245	-0.0220	-0.0271	-0.0248	-0.0781	-0.0319	-0.0194	-0.0265	-0.1193	-0.0369	-0.0248	-0.0271
00	0.1000	-0.0817	-0 1067	-0 1170	-0.1147	-0.1056	-0.0639	-0 1013	-0.0699	-0.1055	-0.0570	-0 0401	-0.1056	-0.1147
00	1.1000	-0.7293	-0.9059	-0.9103	-0.9012	-0.9068	-0.8549	-0.8974	-0 8805	-0 8827	-0.8598	-0 8843	-0 9068	-0.9012
00	2.1000	-0.7063	-0 8792	-0.8485	-0.8381	-0.8692	-0.8739	-0 8513	-0.8932	-0 8490	-0.8827	-0.8871	-0.8692	-0.8381
00	3 1000	-0.0340	-0.0558	-0.0541	-0.0675	-0.0619	-0.0668	-0.0667	-0.0315	-0 0659	-0.0134	0.0760	-0 0619	-0.0675

)00	4.1000	0.6696	0.8548	0.7712	0.7999	0.8681	0.8542	0.8159	0.8107	0.7834	0.8279	0.8126	0.8681	0.7999
)00	5.1000	0.7576	0.9110	0.8705	0.8594	0.9257	0.8162	0.8949	0.8192	0.8841	0.7690	0.7282	0.9257	0 8594
)00	6.1000	0.1491	0.1607	0 1602	0.1556	0.1493	0.1143	0.1584	0.1443	0.1522	0.0331	0.1766	0.1493	0.1556
)00	0.1000	-0.0924	-0.1420	-0.1341	-0.1276	-0.1488	-0.2649	-0.1425	-0.1961	-0.1441	-0.2462	-0 2601	-0 1488	-0.1276
)00	1.1000	-0.8251	-0.9132	-0.9392	-0.9714	-0.9159	-0.8649	-0.9480	-0.9830	-0.9925	-0 8529	-0.8594	-0.9159	-0.9714
)00	2.1000	-0.7992	-0.9264	-0.9295	-0.9591	-0.9267	-0.8803	-0.9525	-0.9683	-0.9778	-0.8789	-0.8971	-0.9267	-0 9591
)00	3.1000	-0.0385	-0.0774	-0.0752	-0.0699	-0.0733	-0.0877	-0.0698	-0.0925	-0 0614	-0.1363	-0.1686	-0.0733	-0.0699
)00	4.1000	0.7576	0.9160	0.9149	0.9104	0.8637	0.8278	0.8950	0.8669	0 8831	0.8108	0.8046	0.8637	0.9104
)00	5.1000	0.8571	0.9034	0.9374	0.9499	0.8862	0.8484	0.9165	1.0023	0.9881	0.8198	0.8013	0.8862	0.9499
)00	6.1000	0.1686	0 1854	0.1871	0.1830	0.2056	0.2919	0.1887	0.2441	0.1917	0.2600	0.3560	0.2056	0.1830
)00	0.1000	-0.0182	-0.0308	-0.0481	-0.0376	-0.0263	0.0371	-0.0344	-0.0121	-0.0287	0.0288	0.0367	-0.0263	-0 0376
)00	1.1000	-0.1623	-0.2247	-0.2074	-0.2092	-0.2245	-0.2901	-0.2169	-0.2489	-0.2279	-0.2848	-0.2680	-0.2245	-0 2092
)00	2.1000	-0.1572	-0.2017	-0.1849	-0.2089	-0.2034	-0.1711	-0.2064	-0.1667	-0 1938	-0.1800	-0.1657	-0 2034	-0.2089
)00	3.1000	-0.0076	-0.0356	-0.0650	-0.0322	-0 0257	-0.0179	-0.0242	-0.0200	-0.0310	-0.0567	-0.0064	-0.0257	-0 0322
)00	4.1000	0.1491	0.1669	0.1952	0.1551	0.1512	0.1262	0.1527	0.1314	0 1516	0.1233	0.2075	0.1512	0.1551
)00	5.1000	0.1686	0.1827	0.1637	0.1799	0.2133	0.2626	0.1917	0 2410	0.1943	0.2314	0.2767	0.2133	0.1799
)00	6.1000	0.0332	0.0173	0.0234	0.0203	0.0020	-0.0427	0.0151	-0.0421	0.0126	-0.0772	0.0485	0.0020	0.0203

ere

e 1 = both layer having sigmoid(beta=0.5) activation in Hessian matrix method
e 2 = both layer having sigmoid(beta=0 9) activation in Hessian matrix method
e 3 = both layer having sigmoid(beta=0.5) activation & omega=1.5 in Iterative pruning method
e 4 = both layer having sigmoid(beta=0.9) activation & omega=0.5 in Iterative pruning method
e 5 = both layer having sigmoid(beta=0.5) activation & lamda=5e-3 in weight decay method-1
e 6 = both layer having sigmoid(beta=0.9) activation & lamda=1e-3 in weight decay method-1
e 7 = both layer having sigmoid(beta=0.5) activation & lamda=1e-3 in weight decay method-2
e 8 = both layer having sigmoid(beta=0.9) activation & lamda=1e-3 in weight decay method-2
e 9 = both layer having sigmoid(beta=0.9) activation & lamda=1e-5,w0=1.05 in weight elimination method
e 10 = both layer having sigmoid(beta=0.5) activation & lamda=5e-4,w0=0.95 in weight elimination method
e 11 = both layer having sigmoid(beta=0.5) activation for simple convergence
e 12 = both layer having sigmoid(beta=0.9) activation for simple convergence

Table 4.38 FOR ARCHITECTURES HAVING 40 HIDDEN NEURONS

patterns dians)	output patterns	case 1	case 2	case 3	case 4	case 5	case 6	case 7	case 8	case 9	case 10	case 11	case 12
0 0.1000	0.0100	-0.0079	-0.0123	-0.0122	-0.0116	-0.0739	-0.0562	-0.0188	-0.0501	-0.0748	-0.1417	-0.0122	-0.0116
0 1.1000	0.0890	0.0873	0.0996	0.0983	0.0823	0.1763	0.1310	0.1026	0.1380	0.2067	0.1621	0.0983	0.0823
0 2.1000	0.0862	0.0867	0.0741	0.0803	0.0786	0.0308	0.0624	0.0722	0.0519	0.0274	-0.0075	0.0803	0.0786
0 3.1000	0.0042	-0.0188	-0.0037	-0.0212	-0.0196	-0.0090	-0.0085	-0.0104	-0.0045	0.0131	0.0406	-0.0212	-0.0196
0 4.1000	-0.0817	-0.1146	-0.1284	-0.0962	-0.1207	-0.0736	-0.1047	-0.1094	-0.0830	-0.0137	0.1055	-0.0962	-0.1207
0 5.1000	-0.0924	-0.1286	-0.1267	-0.1576	-0.1321	-0.2031	-0.1718	-0.1427	-0.1934	-0 2027	-0.0345	-0 1576	-0.1321
0 6.1000	-0.0182	-0.0400	-0.0363	-0.0222	-0.0423	0.0201	0.0043	-0.0300	0.0087	0.0447	0 1113	-0.0222	-0.0423
0 0.1000	0.0890	0.0889	0.0981	0.0958	0.0836	0.1529	0.1492	0.0999	0.1380	0.1389	0.0268	0.0958	0.0836
0 1.1000	0.7943	0.8999	0.8756	0.8905	0.9246	0.8667	0.8644	0.9160	0.9173	0.8551	0.7661	0 8905	0.9246
0 2.1000	0.7693	0.9089	0.8840	0.9024	0.9018	0.8710	0.8849	0.9002	0.8773	0.8440	0.7439	0.9024	0.9018
0 3.1000	0.0371	0.0270	0.0348	0.0350	0.0219	0.0131	0.0207	0.0230	0.0217	0.0342	0.0465	0.0350	0.0219
0 4.1000	-0.7293	-0.9258	-0.9363	-0.9169	-0.9146	-0.8626	-0 8572	-0.8913	-0.8715	-0.8371	-0.6883	-0.9169	-0.9146
0 5.1000	-0.8251	-0.9456	-0.9301	-0.9233	-0.9623	-0.8871	-0.8919	-0.9888	-0.9947	-0.8756	-0 7601	-0.9233	-0.9623
0 6.1000	-0.1623	-0.2152	-0.2294	-0.2299	-0.2174	-0.2909	-0.2974	-0.2271	-0 2615	-0 2915	-0.2698	-0.2299	-0.2174
0 0.1000	0.0862	0.0838	0.0749	0.0826	0.0788	0.0490	0.0623	0.0766	0.0536	0.0355	-0.0756	0.0826	0.0788
0 1.1000	0.7693	0.8982	0.9045	0.8920	0.8957	0.8690	0.8667	0.8947	0.8766	0.8435	0.7139	0 8920	0 8957
0 2.1000	0.7451	0.8779	0.8932	0.8514	0 8480	0.8819	0.8741	0 8673	0.8787	0 8709	0.7955	0.8514	0 8480
0 3.1000	0.0359	0.0231	0.0147	0.0209	0.0237	0.0311	0.0481	0.0285	0.0289	-0.0046	-0.0670	0 0209	0.0237
0 4.1000	-0.7063	-0.8775	-0.8984	-0.8803	-0.8807	-0.8874	-0.8880	-0 8594	-0.8693	-0 8755	-0.8361	-0.8803	-0 8807
0 5.1000	-0.7992	-0.9292	-0.9080	-0.9318	-0.9341	-0.8961	-0.8921	-0.9672	-0.9664	-0.8656	-0 7862	-0 9318	-0.9341
0 6.1000	-0.1572	-0.2080	-0.1894	-0.1939	-0.2094	-0.1695	-0.1477	-0.1967	-0.1736	-0.1462	-0.1526	-0.1939	-0.2094
0 0.1000	0.0042	-0.0145	-0.0095	-0.0142	-0.0175	-0.0185	-0.0365	-0.0131	-0.0082	-0.0014	-0.0871	-0.0142	-0.0175
0 1.1000	0.0371	0.0262	0.0277	0.0264	0.0217	0.0318	0.0323	0.0253	0.0262	0.0195	-0 0851	0 0264	0 0217
0 2.1000	0.0359	0.0208	0 0286	0.0339	0.0228	0.0297	0.0164	0.0253	0 0256	0.0238	-0.0884	0.0339	0 0228
0 3.1000	0.0017	-0.0147	-0.0137	-0.0123	-0.0248	-0.0319	-0.0497	-0 0246	-0 0102	-0.0471	-0.1607	-0 0123	-0.0248
0 4.1000	-0.0340	-0.0608	-0.0545	-0.0593	-0.0596	-0.0638	-0.0807	-0.0597	-0.0572	-0.0978	-0.2778	-0 0593	-0.0596
0 5.1000	-0 0385	-0.0632	-0.0760	-0.0759	-0.0698	-0.0654	-0.0711	-0.0745	-0.0683	-0.0830	-0.2448	-0.0759	-0 0698
0 6.1000	-0.0076	-0 0287	-0 0558	-0.0369	-0.0304	-0.0396	-0.0329	-0 0289	-0.0254	-0.0369	-0.0155	-0 0369	-0.0304
0 0.1000	-0.0817	-0 1155	-0 1184	-0.1108	-0.1188	-0.0849	-0 0774	-0.1103	-0.0844	-0 0689	-0.0820	-0.1108	-0 1188
0 1.1000	-0.7293	-0.8893	-0.9565	-0.9182	-0.8952	-0 8746	-0 8710	-0.8874	-0.8736	-0.8594	-0.7999	-0 9182	-0.8952
0 2.1000	-0.7063	-0 8412	-0.8851	-0.8612	-0.8639	-0.8878	-0.8940	-0 8587	-0.8659	-0.8793	-0.8598	-0 8612	-0.8639
0 3.1000	-0.0340	-0.0672	-0.0668	-0.0733	-0.0611	-0.0648	-0.0368	-0 0564	-0.0618	-0.0623	-0.2146	-0 0733	-0 0611

0	4.1000	0.6696	0.7955	0.8081	0.8383	0.7713	0.8622	0.8810	0.7828	0.7849	0.8329	0.6801	0.8383	0.7713
0	5.1000	0.7576	0.8679	0.8948	0.9127	0.8816	0.8794	0.8793	0.8915	0.8798	0.8254	0.6130	0.9127	0.8816
0	6.1000	0.1491	0.1595	0.1806	0.1687	0.1532	0.1437	0.0918	0.1504	0.1267	0.0918	-0.0166	0.1687	0.1532
0	0.1000	-0.0924	-0.1299	-0.1303	-0.1412	-0.1318	-0.1888	-0.1758	-0.1415	-0.1888	-0.2074	-0.1827	-0.1412	-0.1318
0	1.1000	-0.8251	-0.9739	-0.9580	-0.9317	-0.9707	-0.8870	-0.8900	-0.9896	-0.9964	-0.8844	-0.8376	-0.9317	-0.9707
0	2.1000	-0.7992	-0.9648	-0.9189	-0.9350	-0.9518	-0.8879	-0.9092	-0.9753	-0.9642	-0.8752	-0.8280	-0.9350	-0.9518
0	3.1000	-0.0385	-0.0692	-0.0679	-0.0699	-0.0711	-0.0777	-0.0776	-0.0657	-0.0677	-0.0877	-0.2395	-0.0699	-0.0711
0	4.1000	0.7576	0.9438	0.8735	0.9121	0.9247	0.8578	0.8881	0.8836	0.8712	0.8227	0.6641	0.9121	0.9247
0	5.1000	0.8571	0.9545	0.9354	0.9047	0.9554	0.8678	0.8945	0.9828	0.9858	0.8397	0.7234	0.9047	0.9554
0	6.1000	0.1686	0.1823	0.1702	0.1832	0.1800	0.2327	0.2890	0.1975	0.2300	0.2399	-0.0461	0.1832	0.1800
0	0.1000	-0.0182	-0.0395	-0.0397	-0.0323	-0.0420	0.0148	-0.0127	-0.0303	0.0044	0.0261	-0.0003	-0.0323	-0.0420
0	1.1000	-0.1623	-0.2142	-0.2144	-0.2252	-0.2156	-0.3005	-0.2513	-0.2265	-0.2556	-0.3120	-0.3222	-0.2252	-0.2156
0	2.1000	-0.1572	-0.2008	-0.2073	-0.1944	-0.2071	-0.1542	-0.1829	-0.1959	-0.1752	-0.1407	-0.1597	-0.1944	-0.2071
0	3.1000	-0.0076	-0.0341	-0.0313	-0.0425	-0.0284	-0.0346	-0.0509	-0.0303	-0.0293	-0.0269	-0.0547	-0.0425	-0.0284
0	4.1000	0.1491	0.1632	0.1660	0.1753	0.1549	0.1162	0.1539	0.1538	0.1309	0.1160	0.0047	0.1753	0.1549
0	5.1000	0.1686	0.1800	0.1781	0.1758	0.1812	0.2766	0.2325	0.1947	0.2335	0.2699	0.1064	0.1758	0.1812
0	6.1000	0.0332	0.0208	0.0242	0.0225	0.0156	-0.0405	-0.0470	0.0097	-0 0213	-0 0507	-0.3216	0.0225	0 0156

·e

1 = both layer having sigmoid(beta=0.9) activation in Hessian matrix method
2 = both layer having sigmoid(beta=0.5) activation in Hessian matrix method
3 = both layer having sigmoid(beta=0.5) activation & omega=1.5 in Iterative pruning method
4 = both layer having sigmoid(beta=0.9) activation & omega=0.5 in Iterative pruning method
5 = both layer having sigmoid(beta=0.9) activation & lamda=5e-3 in weight decay method-1
6 = both layer having sigmoid(beta=0.9) activation & lamda=1e-3 in weight decay method-1
7 = both layer having sigmoid(beta=0.9) activation & lamda=5e-3 in weight decay method-2
8 = both layer having sigmoid(beta=0.5) activation & lamda=1e-3 in weight decay method-2
9 = both layer having sigmoid(beta=0.5) activation & lamda=1e-4,w0=0.95 in weight elimination method
10 = both layer having sigmoid(beta=0.9) activation & lamda=1e-5,w0=1.05 in weight elimination method
11 = both layer having sigmoid(beta=0.5) activation for simple convergence
12 = both layer having sigmoid(beta=0.9) activation for simple convergence

Test Problem-2 - **XOR-CLASSFICATION PROBLEM**

The Exclusive-OR(XOR) problem is a classical mapping problem for a multi-layer perceptron because of its non linearly separable nature. This problem is a two input and one output problem. The data set consists of 4 training patterns. The whole of the data set is trained without dividing it into both training set and validation set. This is done to access the efficiency of the different pruning algorithms. The cost function is taken to be sum-squared error (SSE). The training algorithm is taken to Levenberg-Marquardt algorithms of back-propagation algorithm. The weights are initialized between $\left(\frac{-2.4}{F_i}, \frac{2.4}{F_i}\right)$, where F_i is the fan-in of the ith node for every pruning algorithm. The single hidden layer architecture is chosen to solve the problem for its simplicity. The activation functions for both layers are chosen to be sigmoid. The inputs and outputs are normalized between the –1.0 to 1.0.The results obtained from different pruning algorithms are discussed below –

Hessian matrix method-

This method is tried with two different architectures, of different number of hidden neurons and the activation function used. In first architectures the sigmoid unit is considered in hidden units and linear activation in output layer and in second architectures both layers' activation function are sigmoid type.

The results with 20 hidden neurons are shown in fig.4.10 and the results are summarized in table 4.39 and table 4.40. The results with 30 hidden neurons are shown in fig.4.11 and the results are summarized in table 4.41 and table 4.42.

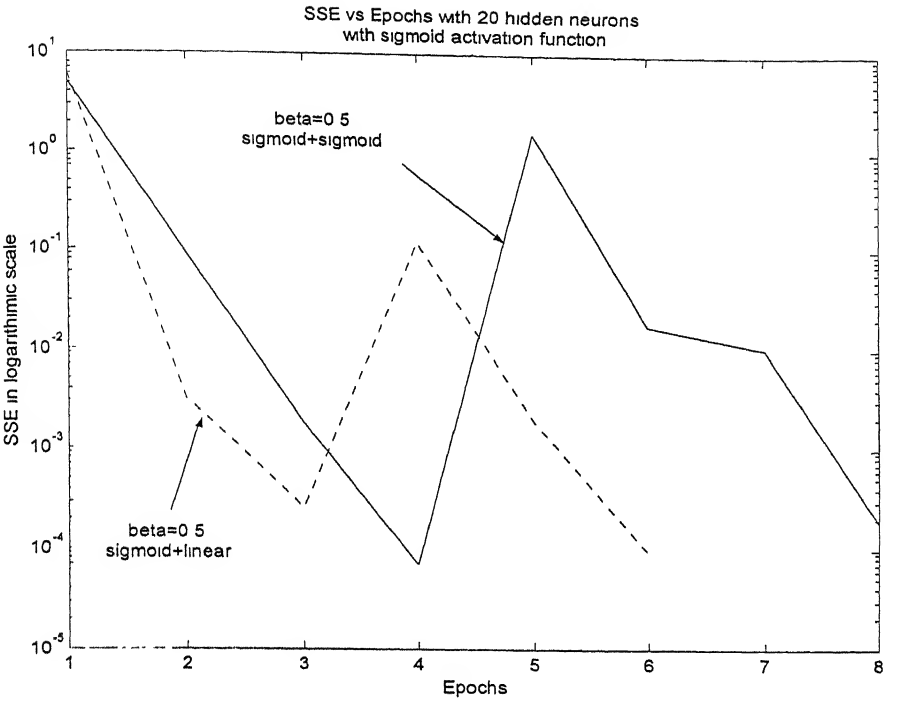


Fig.4.10 Comparison of SSE curves using 20 hidden neurons with different activation functions

Table no.4.39 with beta=0.5, output layer has linear activation

number of hidden neurons - 20 the steepness constant of sigmoid units(hidden layer) - 0.5 threshold value for deletion of weigths - 0.0001 number of hidden units after pruning - 15 training epochs before pruning - 3 training epochs after pruning - 3 error goal for training process - 0.001

Table no. 4.40 beta=0.5, both layers have sigmoid activation

number of hidden neurons - 20 the steepness constant of sigmoid units - 0.5 threshold value for deletion of weigths - 0.0001 number of hidden units after pruning - 5 training epochs before pruning - 4 training epochs after pruning - 4 error goal for training process - 0.001

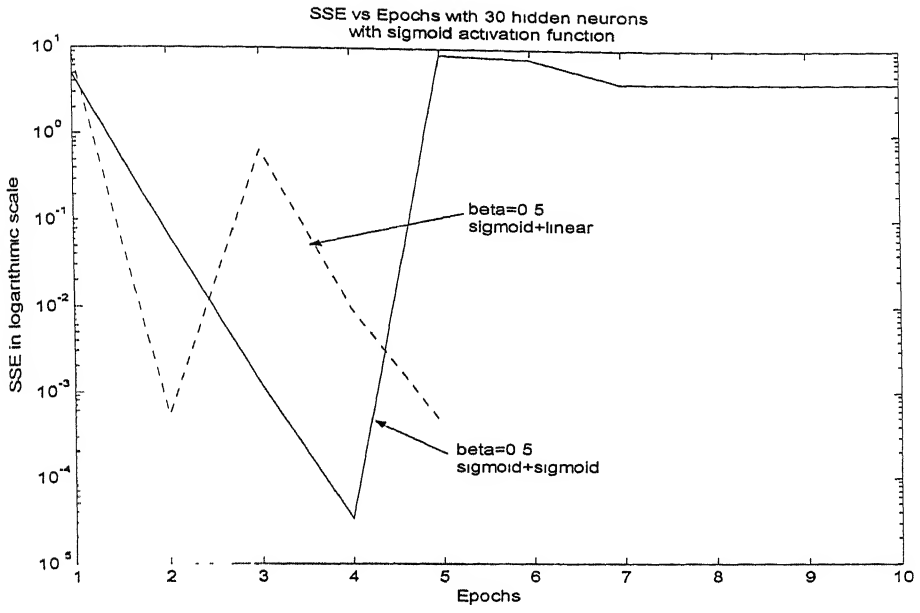


Fig. 4.11 Comparison of SSE curves using 30 hidden neurons with different activation functions

Table no.4.41 beta=0.5, both layers have sigmoid activation

number of hidden neurons - 30
the steepness constant of sigmoid units - 0.5
threshold value for deletion of weigths - 0.001
number of hidden units after pruning - 2
training epochs before pruning - 4
training epochs after pruning - 6
error goal for training process - 0.001

Table no. 4.42 with beta=0.5,output layer has linear activation

number of hidden neurons - 30
the steepness constant of sigmoid units - 0.5
threshold value for deletion of weigths - 0.0001
number of hidden units after pruning - 24
training epochs before pruning - 2
training epochs after pruning - 3
error goal for training process - 0.001

Discussions-

The observed peaks are the increased error just after pruning and again the error value is reduced in the process of retraining. Hence the benefits of this method can be easily pointed out as it automatically compensates the increase in error due to the pruning process. But the number of training epochs after the pruning should be kept as much as near to the training epochs before pruning. It can be easily observed that the value of steepness constant(beta) can affect the convergence during the training period and the number of hidden nodes to be deleted. The number of hidden neurons also plays an important role in deciding the minimum number of hidden nodes after pruning as it affects the convergence speed of training algorithm. In this problem, it is observed when linear activation function is taken at output layer keeping the threshold value for deletion of weights, the normalization range of input and output, number of weight connections between input to hidden and hidden to output layer after deletion is different. All other pruning procedures are started form the same initial weights.

Iterative Pruning Procedure—

The network is trained with the same initial weights that of the Hessian matrix method. After training procedure, the pruning algorithm is applied to for the different values of the pre-conditioning parameter (omega) and it is allowed to proceed until the SSE value does not exceed a specified limit. The results with different hidden neurons, omega are described in table 4.43 and table 4.44 .

Table no. 4.43- Results with 20 hidden units

number of hidden neurons - 20
steepness parameter for sigmoid unit - 0.5
and both layers contains sigmoid units
for omega = 0.5 , the number of hidden neurons
after pruning - 5 ,error at end of pruning = 2.0

for omega = 1.8 , the number of hidden neurons
after pruning - 8 ,error at end pruning = 1.0145

steepness parameter for hidden sigmoid unit - 0.5
and output layer's activation is linear
for omega = 0.5 , the number of hidden neurons
after pruning - 13 ,error at end of pruning = 0.3181

for omega = 1.5 , the number of hidden neurons
after pruning - 14 ,error at end of pruning = 0.3273

Table 4.44- Results with 30 hidden nodes

number of hidden neurons - 30
steepness parameter for sigmoid unit - 0.5
and both layers contains sigmoid units
for omega = 0.5 , the number of hidden neurons
after pruning - 4 ,error at end of pruning = 2.0031

for omega = 1.5 , the number of hidden neurons
after pruning - 5 ,error at end pruning = 0.5703

steepness parameter for hidden sigmoid unit - 0.5
and output layer's activation is linear
for omega = 0.5 , the number of hidden neurons
after pruning - 13 ,error at end of pruning = 0.2734

for omega = 1.5 , the number of hidden neurons
after pruning - 11 ,error at end of pruning = 0.2220

Discussions-

This method only depends on the activation function of the neurons, not the training procedures. Though this method has the advantage of not requiring retraining process, but the results are poorer as compared to Hessian matrix method. The effect of steepness parameter(beta) and the pre-conditioning parameter(omega) can be observed from the above tables. In general, the higher the value of omega, the higher number of hidden units is deleted. This pruning

process shows a drastic change in SSE value during the process of pruning, which is the point to stop pruning.

Penalty function Procedures-

- **Weight decay process-1-**

This pruning process is also tried with different regularization parameter(lamda) and with both 20 and 30 hidden neuron architecture. The error surfaces with different regularization parameter and different activation function are given fig. 4.12 and 4 13. The weight values after the training process are listed in table 4.45,4.46,4.47,4.48,4.49,4.50,4.51,4.52 give an illustration of pruning algorithm. In tables, w1 indicates the input to hidden layer weights, w2 indicates the hidden to output layer weights.

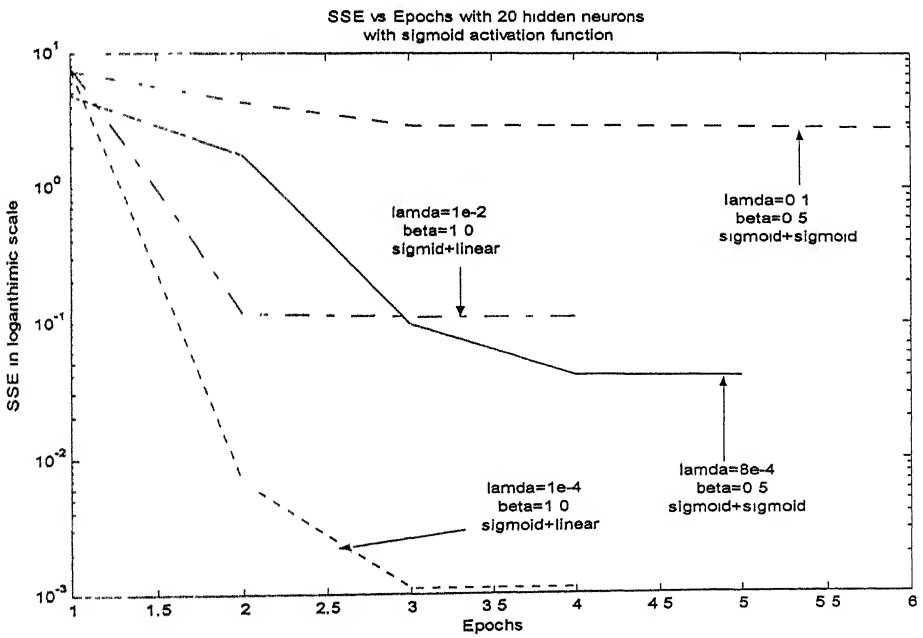


Fig. 4.12.Comparison SSE curves for different lamda values

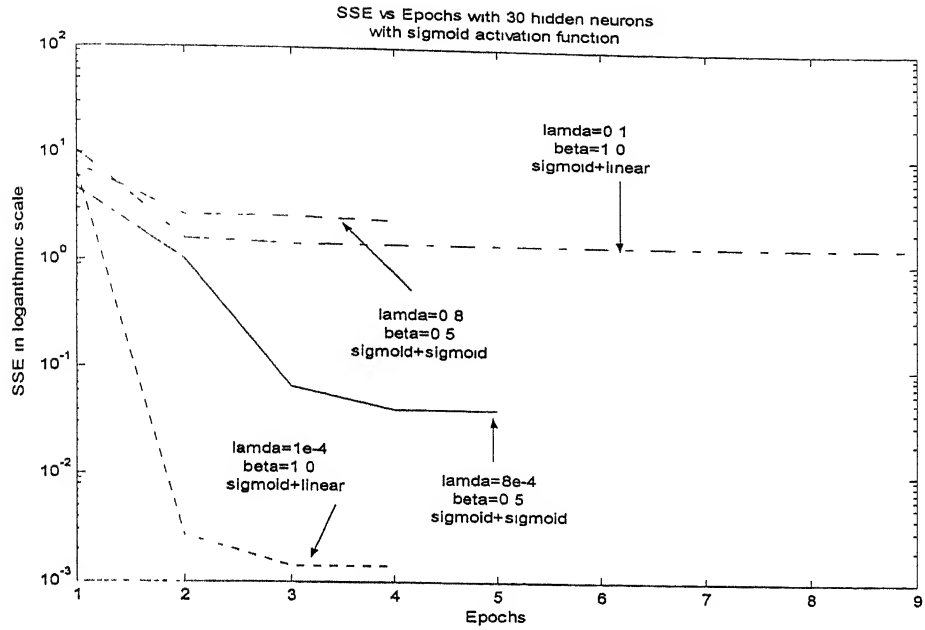


Fig. 4.13 Comparison of SSE curves for different lamda values

For architectures with sigmoid(beta=1.0) activation in hidden layer & linear activation in output layer

Table no.4.45 lamda=1e-4

w1		w2
1.2316	1.3909	1.3168
1.6206	1.7059	1.9047
0.4589	-0.2580	-0.6205
0.5459	0.1146	-0.1664
-0.9065	-1.2050	0.8827
0.9828	-0.4067	-0.6355
-0.2876	0.0283	-0.2490
-1.8693	1.9827	-2.6762
1.2867	1.4091	1.3840
1.2281	-0.8183	-1.0687
-1.2740	1.3666	-1.6300
-0.3880	0.5185	-0.6031
-2.2959	-2.2051	3.2632
0.5456	-0.9265	-0.6467
1.2572	-1.0621	-1.4029
-0.3195	-1.1223	0.2562
-1.0676	-1.0280	1.0463
-0.3672	-0.4261	0.1411
1.5387	1.6299	1.7831
1.2316	-1.1270	-1.5980

Table no. 4.46 lamda=1e-3

w1		w2
0.8434	1.2344	0.8223
1.3149	1.2456	1.2179
0.3555	-0.0767	-0.3675
0.5358	0.1504	-0.0814
-0.7354	-1.1375	0.5443
0.9470	-0.1620	-0.4783
-0.3275	-0.1633	-0.1720
-1.4521	1.4800	-1.7932
1.0664	1.1648	0.9566
1.1178	-0.3304	-0.6698
-0.9540	0.9168	-0.9338
-0.1693	0.4309	-0.4112
-1.7803	-1.7202	2.2852
0.2926	-0.8586	-0.3709
1.0568	-0.6211	-0.9734
-0.2467	-1.1027	-0.0094
-0.8188	-0.8446	0.5400
-0.3687	-0.4335	-0.0594
1.1951	1.3524	1.2807
0.8339	-0.7105	-1.1109

For architectures with sigmoid(beta=0 5) activation in both layers

Table no.4.47 lamda=8e-4

w1		w2
0.5224	1.0414	0.2901
1.0927	0.7936	0.3784
0.3621	-0.0283	-0.2236
0.5642	0.2534	-0.1075
-0.3771	-1.0609	0.1607
0.9218	0.0857	-0.2030
-0.3517	-0.0885	-0.1620
-1.0563	0.8848	-0.5557
0.5757	0.8911	0.3649
1.0984	-0.0077	-0.3106
-0.8161	0.6621	-0.3674
-0.2052	0.3763	-0.2482
-1.0098	-1.1972	0.6524
-0.0221	-0.8787	-0.2054
0.8917	-0.1258	-0.4147
-0.2295	-1.1085	0.0142
-0.5759	-0.6450	0.2333
-0.3467	-0.4115	-0.0131
0.6416	0.9961	0.4535
0.3915	-0.4668	-0.5302

Table no. 4.48 lamda=0.1

w1		w2
0.5253	1.0385	0.2864
1.0958	0.7962	0.3749
0.3649	-0.0311	-0.2189
0.5671	0.2565	-0.1094
-0.3743	-1.0640	0.1651
0.9188	0.0830	-0.2048
-0.3488	-0.0913	-0.1634
-1.0533	0.8821	-0.5577
0.5728	0.8937	0.3673
1.0952	-0.0102	-0.3062
-0.8131	0.6593	-0.3635
-0.2023	0.3735	-0.2497
-1.0124	-1.2006	0.6559
-0.0252	-0.8814	-0.2062
0.8887	-0.1280	-0.4159
-0.2266	-1.1114	0.0132
-0.5731	-0.6423	0.2315
-0.3438	-0.4086	-0.0145
0.6444	0.9986	0.4556
0.3883	-0.4689	-0.5304

For architectures with sigmoid(beta=0.5) activation in both layers

Table 4.49 with lamda=8e-4

w1		w2
0.4674	0.1229	0.1812
1.5253	-1.4650	-2.3056
-0.0618	-0.0347	0.1436
-0.9465	-0.5236	0.8268
-0.0672	-0.5058	0.2658
1.2223	-0.8906	-1.2814
-0.5579	0.7723	-0.8153
-0.9413	-1.2019	1.6381
1.1242	-1.2091	-1.4890
-0.6680	0.7350	-0.8647
-1.0783	1.0729	-1.5313
0.4911	0.4068	0.3678
-0.9906	1.1088	-1.5325
-0.1099	-0.3049	0.2243
0.9962	-0.7560	-1.0037
0.9642	-0.4672	-0.6076
0.6255	0.6197	0.6856
0.8955	-0.9608	-1.0592
1.3960	1.2366	1.8990
-0.2527	-0.1289	0.2356
-1.1149	1.0249	-1.4669
0.9441	1.2538	1.2620
1.0072	-0.5149	-0.6845
-1.0675	1.0223	-1.4518
-1.1569	0.6999	-0.9505
-0.1194	0.4581	-0.1943
-1.6704	-1.7356	3.1890
0.4017	0.3956	0.3186
1.5062	1.4620	2.2962
-0.1351	0.2997	-0.1484

Table 4.50 with lamda=0.8

w1		w2
0.3023	-0.0587	-0.0421
1.0842	-1.0541	-1.3752
-0.2214	-0.1896	-0.0491
-0.7495	-0.4755	0.3926
0.1122	-0.6590	0.0786
1.2720	-0.5146	-0.7802
-0.2403	0.4961	-0.3707
-0.4653	-1.1893	0.8872
0.7697	-0.9353	-0.8638
-0.4041	0.7564	-0.5346
-1.0433	0.9547	-0.9248
0.6300	0.2328	0.1826
-0.5728	0.7169	-0.8419
-0.2654	-0.4640	0.0861
1.0442	-0.7371	-0.5703
1.0810	-0.1978	-0.3188
0.3905	0.7081	0.3268
0.8854	-1.0246	-0.7104
1.3959	1.1245	1.1837
-0.4064	-0.2657	0.0167
-0.8155	0.8737	-0.8474
0.5465	1.0487	0.6628
0.8005	-0.2220	-0.3560
-1.0561	0.9020	-0.8730
-0.9527	0.3238	-0.5407
-0.2335	0.2797	0.0016
-1.1251	-1.5091	1.9506
0.5429	0.2277	0.1602
1.1184	1.3638	1.4185
0.0514	0.1175	0.0510

For architectures with sigmoid(beta=1.0) activation in hidden layer & linear activation in output layer

Table 4.51 with lamda=1e-4

w1		w2
0.4646	0.0965	0.0211
1.0551	-1.0429	-0.3521
-0.0557	-0.0188	0.0223
-0.8922	-0.2085	0.1227
-0.0237	-0.4938	-0.0359
1.0695	-0.5047	-0.1982
-0.3289	0.6171	-0.1889
-0.4156	-0.9687	0.2190
0.6747	-1.0137	-0.2699
-0.4845	0.5274	-0.1494
-0.7749	0.6402	-0.2452
0.4512	0.3716	-0.0263
-0.5813	0.7586	-0.2950
-0.0880	-0.2936	-0.0579
0.8287	-0.4726	-0.2063
0.9113	-0.2914	-0.0737
0.4932	0.5067	0.1211
0.6123	-0.7958	-0.1760
1.1386	0.7664	0.2273
-0.2373	-0.0868	0.0254
-0.8866	0.5479	-0.2504
0.5532	1.1843	0.1356
0.9553	-0.3115	-0.0833
-0.7953	0.5728	-0.2695
-1.0985	0.3477	-0.1106
-0.0566	0.4377	-0.0937
-0.9785	-1.1829	0.3873
0.3593	0.3693	0.0033
1.0899	1.0719	0.2488
-0.1063	0.2763	-0.0959

Table 4.52 with lamda=0.1

w1		w2
0.4681	0.1001	0.0247
1.0575	-1.0461	-0.3529
-0.0521	-0.0152	0.0264
-0.8964	-0.2039	0.1264
-0.0273	-0.4902	-0.0391
1.0643	-0.5076	-0.1964
-0.3315	0.6193	-0.1908
-0.4178	-0.9659	0.2216
0.6696	-1.0166	-0.2712
-0.4872	0.5223	-0.1473
-0.7700	0.6346	-0.2459
0.4478	0.3755	-0.0307
-0.5835	0.7601	-0.2959
-0.0844	-0.2901	-0.0612
0.8235	-0.4679	-0.2083
0.9064	-0.2945	-0.0727
0.4968	0.5031	0.1236
0.6072	-0.7913	-0.1738
1.1356	0.7623	0.2277
-0.2337	-0.0831	0.0295
-0.8891	0.5424	-0.2512
0.5562	1.1886	0.1369
0.9575	-0.3146	-0.0821
-0.7904	0.5672	-0.2703
-1.1015	0.3503	-0.1081
-0.0527	0.4410	-0.0967
-0.9812	-1.1798	0.3888
0.3557	0.3730	0.0025
1.0936	1.0684	0.2489
-0.1097	0.2796	-0.0988

Discussions-

From the above weight matrix table, it is evident that this method does not produce any significant reduction in weight values. In the weight table, we find some of the weights are reduced to minimum i.e. less than 0.001 and some of the weights are saturated around 1.5 to 2.5. The error surface is very much sensitive to the regularization parameter (lamda). The increase in the value of lamda saturates the ANN and it does not converge to desired minimum value.

- **Weight decay method-2 –**

This procedure is also tried with 20 and 30 hidden neurons and different regularization parameter. The SSE surfaces with 20 and 30 hidden neurons are shown in fig 4.14 and fig. 4.15 respectively. The weight values after the training process are listed in table 4.53,4.54,4.55,4.56,4.57,4.58,4.59,4.60 give an illustration of the pruning algorithm. In tables, w1 indicates the input to hidden layer weights, w2 indicates the hidden to output layer weights.

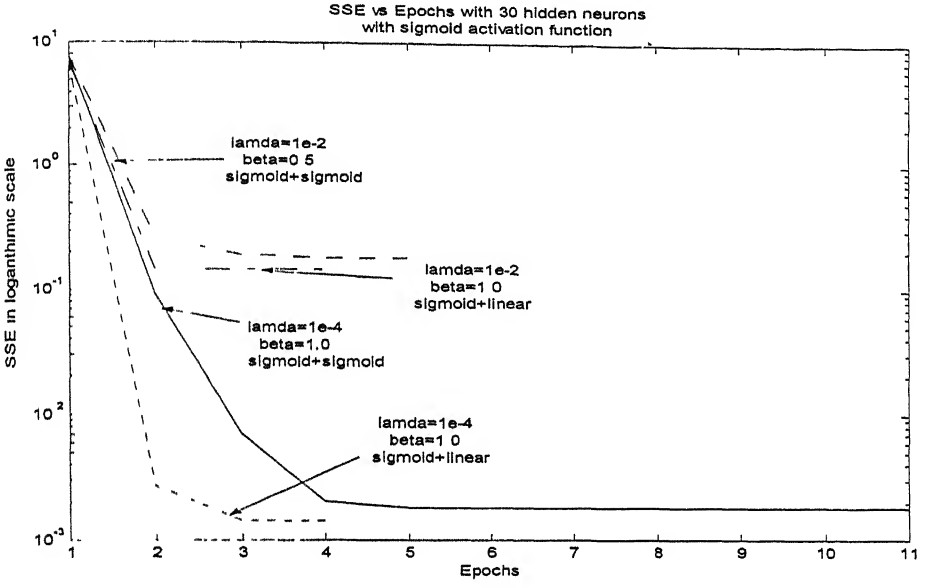


Fig. 4.14 Comparison of SSE curves for different lamda values

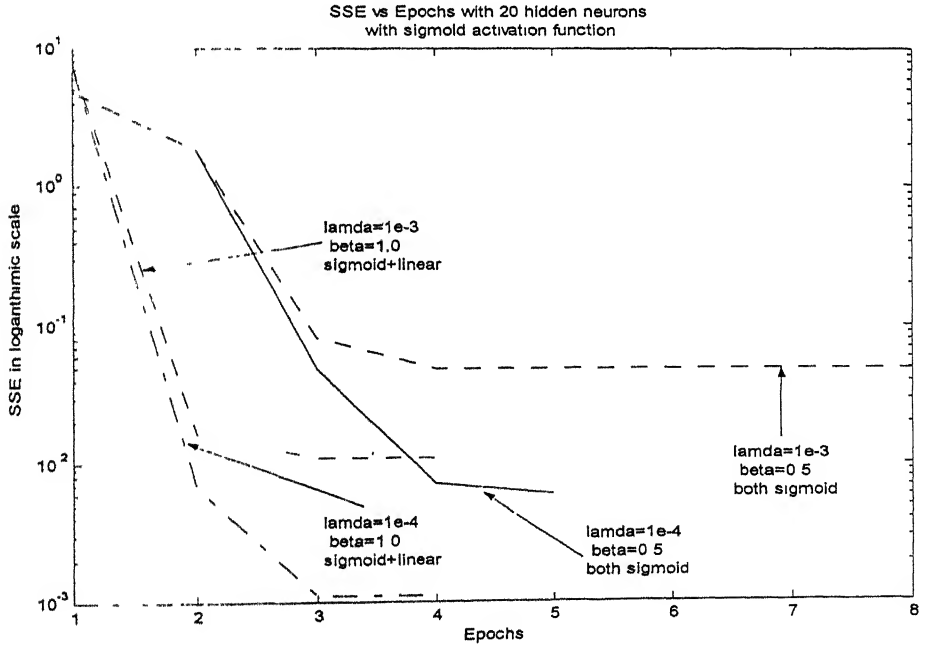


Fig. 4.15 Comparison of SSE curves for different lamda values

For architectures sigmoid(beta=0.5) activation function in both layers

Table 4.53 lamda=1e-4

w1		w2
1.2232	1.3973	1.8673
1.5595	1.6629	2.4943
0.4903	-0.3024	-0.8602
0.5561	0.1739	-0.1290
-0.8299	-1.1737	1.0863
0.9737	-0.3664	-0.8047
-0.2954	-0.0090	-0.2551
-1.7232	1.8438	-3.2936
1.1778	1.3230	1.7637
1.2475	-0.8532	-1.4915
-1.2318	1.3085	-2.1004
-0.3472	0.4719	-0.7143
-2.1933	-2.0529	4.1132
0.4178	-0.9040	-0.7224
1.1534	-0.9181	-1.6731
-0.3194	-1.1213	0.3382
-1.0374	-0.9966	1.4020
-0.3653	-0.4312	0.2147
1.4137	1.5254	2.2581
1.1418	-1.0518	-2.0365

Table 4.54 lamda=1e-3

w1		w2
1.1097	1.3324	1.6653
1.4335	1.5653	2.2071
0.4553	-0.2617	-0.7888
0.5504	0.1800	-0.1101
-0.7454	-1.1276	0.9290
0.9583	-0.2943	-0.7017
-0.3010	-0.0221	-0.2301
-1.5673	1.7169	-2.8690
1.0805	1.2442	1.5763
1.2058	-0.7409	-1.2816
-1.1285	1.1951	-1.8249
-0.3165	0.4498	-0.6454
-2.0911	-1.8668	3.6379
0.3523	-0.8902	-0.6227
1.0857	-0.8068	-1.4579
-0.2956	-1.1146	0.2818
-0.9586	-0.9205	1.2304
-0.3594	-0.4288	0.1936
1.2937	1.4332	2.0134
1.0271	-0.9321	-1.8007

For architectures with sigmoid(beta=1.0) activation
in hidden layer & linear activation in output layer

Table 4.55 with lamda=1e-4

w1		w2
0.5224	1.0414	0.2901
1.0927	0.7936	0.3785
0.3621	-0.0283	-0.2236
0.5642	0.2533	-0.1075
-0.3771	-1.0609	0.1607
0.9219	0.0857	-0.2030
-0.3517	-0.0885	-0.1620
-1.0563	0.8848	-0.5557
0.5757	0.8911	0.3649
1.0984	-0.0076	-0.3106
-0.8161	0.6621	-0.3674
-0.2052	0.3763	-0.2481
-1.0098	-1.1972	0.6524
-0.0220	-0.8787	-0.2054
0.8917	-0.1258	-0.4147
-0.2295	-1.1085	0.0142
-0.5759	-0.6450	0.2333
-0.3467	-0.4115	-0.0131
0.6416	0.9961	0.4535
0.3915	-0.4667	-0.5302

Table 4.56 with lamda=1e-3

w1		w2
0.5225	1.0412	0.2901
1.0930	0.7938	0.3784
0.3624	-0.0284	-0.2236
0.5644	0.2534	-0.1075
-0.3770	-1.0612	0.1607
0.9218	0.0854	-0.2030
-0.3515	-0.0885	-0.1620
-1.0562	0.8846	-0.5557
0.5756	0.8913	0.3649
1.0981	-0.0076	-0.3106
-0.8159	0.6620	-0.3673
-0.2052	0.3760	-0.2481
-1.0099	-1.1974	0.6524
-0.0221	-0.8789	-0.2054
0.8915	-0.1258	-0.4147
-0.2293	-1.1086	0.0142
-0.5758	-0.6448	0.2333
-0.3465	-0.4112	-0.0130
0.6419	0.9962	0.4534
0.3915	-0.4668	-0.5302

For architectures sigmoid(beta=0.5) activation in both layers

Table 4.57 with lamda=1e-4

w1		w2
0.4649	0.0978	0.0551
1.0966	-1.0741	-0.9187
-0.0558	-0.0210	0.0564
-0.8946	-0.2266	0.3132
-0.0246	-0.4936	0.0704
1.0770	-0.5246	-0.5148
-0.3433	0.6289	-0.3772
-0.4454	-0.9806	0.6339
0.7108	-1.0295	-0.5934
-0.4931	0.5381	-0.3664
-0.7905	0.6674	-0.6192
0.4504	0.3721	0.0707
-0.6091	0.7866	-0.6488
-0.0878	-0.2922	0.0370
0.8401	-0.4891	-0.4266
0.9097	-0.2951	-0.2406
0.5032	0.5126	0.2512
0.6269	-0.8021	-0.4166
1.1601	0.8007	0.6862
-0.2378	-0.0899	0.0881
-0.9009	0.5816	-0.5930
0.5806	1.1904	0.4200
0.9536	-0.3161	-0.2702
-0.8129	0.6064	-0.5995
-1.0978	0.3643	-0.3618
-0.0613	0.4406	-0.1395
-1.0223	-1.2171	1.2448
0.3603	0.3702	0.0752
1.1216	1.0982	0.8290
-0.1091	0.2798	-0.1226

Table 4.58 with lamda=1e-2

w1		w2
0.4649	0.0978	0.0549
1.0960	-1.0737	-0.9168
-0.0558	-0.0210	0.0562
-0.8946	-0.2262	0.3125
-0.0245	-0.4936	0.0701
1.0768	-0.5241	-0.5137
-0.3430	0.6286	-0.3766
-0.4448	-0.9803	0.6324
0.7102	-1.0293	-0.5922
-0.4929	0.5378	-0.3657
-0.7902	0.6668	-0.6178
0.4504	0.3721	0.0703
-0.6086	0.7861	-0.6476
-0.0878	-0.2923	0.0367
0.8399	-0.4888	-0.4258
0.9097	-0.2949	-0.2401
0.5030	0.5125	0.2506
0.6265	-0.8019	-0.4157
1.1599	0.8002	0.6845
-0.2378	-0.0898	0.0879
-0.9006	0.5809	-0.5918
0.5801	1.1904	0.4189
0.9535	-0.3159	-0.2696
-0.8126	0.6059	-0.5983
-1.0977	0.3638	-0.3609
-0.0612	0.4405	-0.1394
-1.0214	-1.2165	1.2420
0.3603	0.3701	0.0749
1.1211	1.0979	0.8269
-0.1091	0.2798	-0.1226

For architectures with sigmoid(beta=1.0) activation in hidden layer & linear activation in output layer

Table 4.59 with lamda=1e-4

w1		w2
0.4646	0.0965	0.0211
1.0551	-1.0429	-0.3521
-0.0557	-0.0188	0.0223
-0.8922	-0.2085	0.1227
-0.0237	-0.4938	-0.0359
1.0695	-0.5047	-0.1982
-0.3289	0.6171	-0.1889
-0.4156	-0.9687	0.2190
0.6747	-1.0137	-0.2699
-0.4845	0.5274	-0.1494
-0.7749	0.6402	-0.2452
0.4512	0.3716	-0.0263
-0.5813	0.7586	-0.2950
-0.0880	-0.2936	-0.0579
0.8287	-0.4726	-0.2063
0.9114	-0.2914	-0.0737
0.4932	0.5067	0.1211
0.6123	-0.7958	-0.1760
1.1386	0.7664	0.2273
-0.2373	-0.0868	0.0254
-0.8866	0.5479	-0.2504
0.5532	1.1843	0.1356
0.9553	-0.3115	-0.0833
-0.7953	0.5728	-0.2695
-1.0985	0.3477	-0.1106
-0.0566	0.4377	-0.0937
-0.9785	-1.1829	0.3873
0.3593	0.3693	0.0033
1.0899	1.0719	0.2488
-0.1063	0.2763	-0.0959

Table 4.60 with lamda=1e-2

w1		w2
0.4648	0.0969	0.0211
1.0552	-1.0430	-0.3521
-0.0553	-0.0186	0.0223
-0.8926	-0.2083	0.1228
-0.0237	-0.4935	-0.0359
1.0694	-0.5050	-0.1982
-0.3292	0.6175	-0.1889
-0.4157	-0.9684	0.2190
0.6747	-1.0137	-0.2699
-0.4847	0.5274	-0.1494
-0.7745	0.6399	-0.2452
0.4510	0.3718	-0.0263
-0.5814	0.7588	-0.2951
-0.0878	-0.2932	-0.0580
0.8285	-0.4722	-0.2063
0.9110	-0.2915	-0.0737
0.4935	0.5064	0.1211
0.6119	-0.7956	-0.1760
1.1384	0.7660	0.2273
-0.2371	-0.0867	0.0254
-0.8869	0.5479	-0.2504
0.5534	1.1844	0.1356
0.9555	-0.3119	-0.0833
-0.7952	0.5724	-0.2695
-1.0987	0.3479	-0.1106
-0.0563	0.4379	-0.0937
-0.9785	-1.1825	0.3874
0.3592	0.3697	0.0033
1.0902	1.0719	0.2488
-0.1065	0.2764	-0.0959

Discussions-

From the above weight matrix table, it is evident that this method does not produce any significant reduction in weight values. In the weight table, we findsome of the weights are reduced to minimum i.e. less than 0.001 and some of the weights are saturated around 1.5 to 2.5. The error surface is very much sensitive to the regularization parameter (lamda). The increase in the value of lamda saturates the ANN and it does not converge to desired minimum value.

- **weight elimination method-**

This procedure is also tried with 20 and 30 hidden neurons and different regularization parameter with same initial weights as that of all other methods. The SSE surfaces with 20 and 30 hidden neurons are shown in fig 4.16 and fig. 4.17 respectively. The weight values after the training process are listed in table 4.61,4.62,4.63,4.64,4.65,4.66,4.67 give an illustration of the pruning algorithm. In tables, w1 indicates the input to hidden layer weights, w2 indicates the hidden to output layer weights.

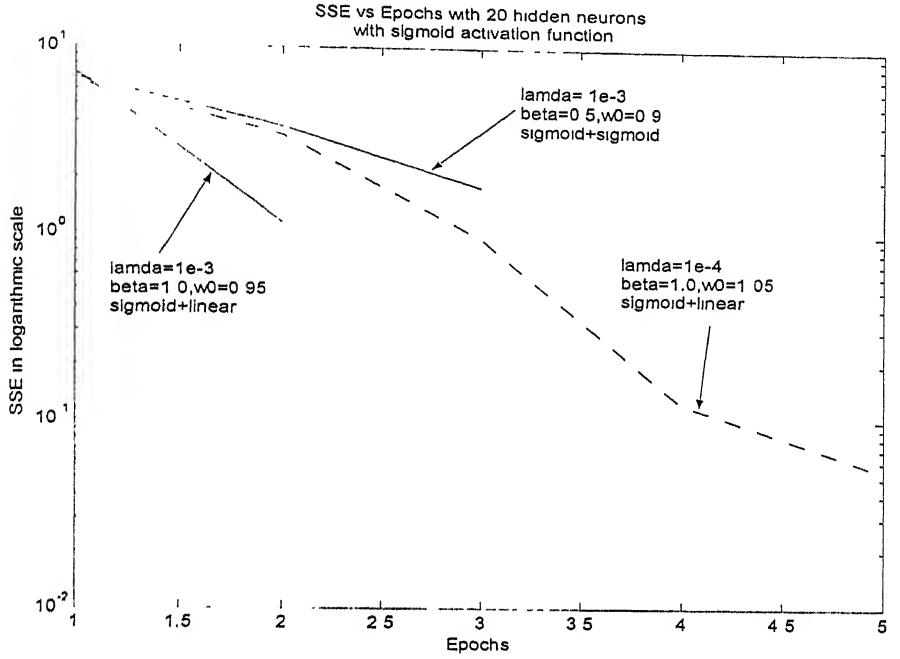


Fig. 4.16 Comparison of SSE curves for different lamda values

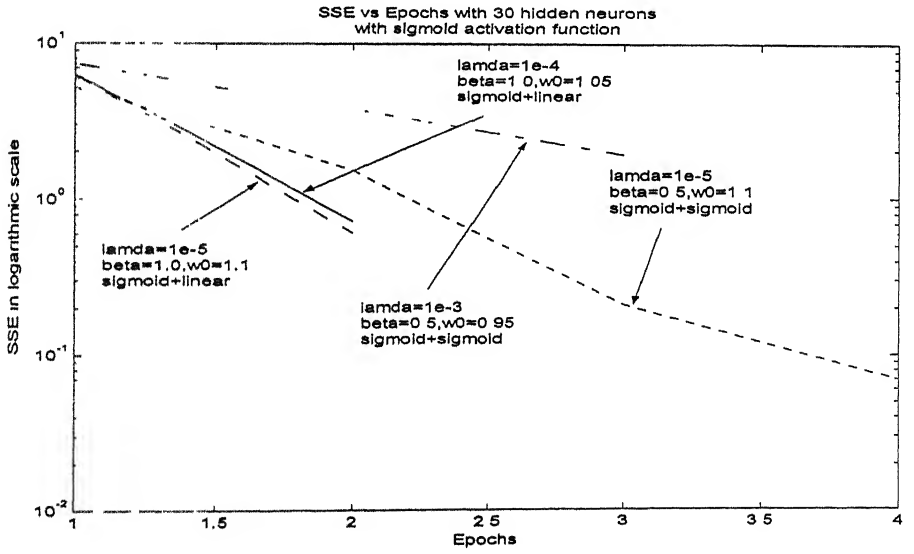


Fig. 4.17 Comparison of SSE curves for different lamda values

For architectures with sigmoid(beta=1.0) activation
in hidden layer & linear activation in output layer

Table no.4.61 lamda=1e-4,w0=1.05

w1		w2
1.2250	1.3980	1.8661
1.5614	1.6629	2.4931
0.4886	-0.3022	-0.8629
0.5554	0.1738	-0.1315
-0.8295	-1.1737	1.0827
0.9726	-0.3656	-0.8055
-0.2957	-0.0087	-0.2599
-1.7250	1.8440	-3.3013
1.1800	1.3229	1.7623
1.2465	-0.8533	-1.4921
-1.2336	1.3087	-2.1073
-0.3481	0.4726	-0.7195
-2.1930	-2.0545	4.1079
0.4174	-0.9033	-0.7248
1.1525	-0.9182	-1.6741
-0.3199	-1.1215	0.3355
-1.0373	-0.9980	1.3977
-0.3656	-0.4318	0.2107
1.4156	1.5254	2.2568
1.1403	-1.0514	-2.0377

Table no.4.62 lamda=1e-3,w0=0.95

w1		w2
0.4942	1.0306	0.5445
1.0929	0.7834	0.6679
0.3495	0.0091	-0.4571
0.5521	0.2103	-0.1350
-0.3753	-1.0606	0.3144
0.9220	0.0801	-0.3576
-0.3407	-0.0764	-0.2417
-1.0553	0.8870	-0.9685
0.6094	0.9176	0.5610
1.0955	0.0502	-0.6093
-0.8083	0.6298	-0.7270
-0.2055	0.3755	-0.4306
-1.0302	-1.1942	1.1396
0.0505	-0.8733	-0.2689
0.8965	-0.2024	-0.6561
-0.2035	-1.1085	0.0792
-0.5548	-0.6314	0.4659
-0.3446	-0.4103	-0 0011
0.6801	1.0219	0.7006
0.4035	-0.4760	-0.9020

For architectures with sigmoid(beta=0.5) activation in both layers

Table no. 4.63 lamda=1e-3,w0=0.9

w1		w2
0.9702	1.2904	1.5118
1.2710	1.5172	1.9570
0.4364	-0.2363	-0.7393
0.5602	0.1874	-0.0969
-0.6654	-1.0878	0.7400
0.9459	-0.2439	-0.6228
-0.3002	-0.0351	-0.2033
-1.3804	1.5425	-2.3911
0.9407	1.2036	1.4315
1.1653	-0.6533	-1.1241
-1.0119	1.0547	-1.5129
-0.2877	0.4102	-0.5592
-2.0596	-1.5670	3.1503
0.2865	-0.8973	-0.5753
1.0139	-0.7184	-1.2984
-0.2643	-1.1084	0.1854
-0.8818	-0.8042	1.0544
-0.3441	-0.4103	0.1641
1.1391	1.3874	1.8106
-0.9055	-0.8481	-1.6322

For architectuers with sigmoid(beta=0.5) activation in both layers

Table no.4.64 lamda=1e-5,w0=1.1

w1		w2
0.4721	0.1374	0.1349
1.6048	-1.5311	-1.8489
-0.0613	-0.0381	0.1052
-0.9582	-0.5482	0.6411
-0.0457	-0.4932	0.1695
1.2360	-0.9267	-1.0125
-0.6027	0.8087	-0.6688
-0.9777	-1.2091	1.2554
1.1999	-1.2529	-1.2160
-0.6811	0.7438	-0.6808
-1.1007	1.1142	-1.2060
0.4728	0.3966	0.2519
-1.0558	1.1749	-1.2346
-0.0846	-0.2869	0.1367
1.0227	-0.7939	-0.8168
0.9501	-0.4531	-0.4629
0.6618	0.6446	0.5432
0.9189	-0.9713	-0.8433
1.4294	1.2960	1.4916
-0.2533	-0.1343	0.1754
-1.1491	1.0858	-1.1713
0.9999	1.2711	0.9914
1.0016	-0.5030	-0.5218
-1.1054	1.0891	-1.1690
-1.1590	0.7161	-0.7286
-0.1351	0.4693	-0.1818
-1.7256	-1.7715	2.4666
0.3952	0.3973	0.2282
1.5567	1.5079	1.7970
-0.1534	0.3144	-0.1487

Table no.4.65 lamda=1e-3,w0=0.95

w1		w2
0.4698	0.1261	0.1112
1.4412	-1.3610	-1.5568
-0.0599	-0.0350	0.0902
-0.9260	-0.4416	0.5286
-0.0268	-0.4901	0.1447
1.1650	-0.7882	-0.8552
-0.5166	0.7448	-0.5831
-0.8008	-1.1117	1.0517
1.0514	-1.1549	-1.0155
-0.6069	0.6653	-0.5924
-0.9796	0.9675	-1.0240
0.4565	0.3785	0.2047
-0.9101	1.0456	-1.0505
-0.0755	-0.2829	0.1125
0.9513	-0.6886	-0.6971
0.9252	-0.3833	-0.3983
0.6101	0.5961	0.4574
0.8125	-0.8971	-0.7140
1.3201	1.1313	1.2393
-0.2479	-0.1218	0.1507
-1.0470	0.9306	-0.9866
0.8618	1.2242	0.8051
0.9734	-0.4232	-0.4470
-0.9928	0.9410	-0.9887
-1.1275	0.5887	-0.6063
-0.1152	0.4614	-0.1738
-1.5007	-1.5686	2.0753
0.3793	0.3839	0.1866
1.4068	1.3452	1.4963
-0.1424	0.3065	-0.1459

For architectures with sigmoid(beta=1.0) activation in hidden layer & linear activation in output layer

Table no.4.66 lamda=1e-4,w0=1.05

w1		w2
0.4622	0.1018	0.0087
1.0678	-1.0409	-0.5940
-0.0596	-0.0257	0.0433
-0.8948	-0.2198	0.3149
-0.0053	-0.4863	0.0517
1.0688	-0.5029	-0.4000
-0.3378	0.6282	-0.3228
-0.4175	-0.9698	0.5496
0.6971	-1.0084	-0.4171
-0.4819	0.5224	-0.2918
-0.7755	0.6439	-0.4570
0.4479	0.3680	-0.0108
-0.5925	0.7767	-0.5144
-0.0710	-0.2817	0.0171
0.8338	-0.4769	-0.3653
0.9011	-0.2749	-0.2171
0.5011	0.5108	0.1458
0.6087	-0.7953	-0.3160
1.1356	0.7763	0.3048
-0.2411	-0.0946	0.0781
-0.8867	0.5636	-0.4286
0.5653	1.1776	0.1455
0.9458	-0.2954	-0.2369
-0.7981	0.5944	-0.4529
-1.1014	0.3321	-0.1999
-0.0615	0.4428	-0.1399
-0.9810	-1.1845	0.8967
0.3588	0.3691	0.0110
1.0929	1.0728	0.3501
-0.1094	0.2835	-0.1209

Table no4.67 lamda=1e-5,w0=1.1

w1		w2
0.4623	0.1018	0.0087
1.0678	-1.0409	-0.5940
-0.0596	-0.0257	0.0433
-0.8948	-0.2198	0.3149
-0.0054	-0.4863	0.0517
1.0688	-0.5029	-0.4000
-0.3378	0.6282	-0.3228
-0.4175	-0.9698	0.5496
0.6971	-1.0084	-0.4171
-0.4819	0.5224	-0.2918
-0.7755	0.6439	-0.4570
0.4478	0.3680	-0.0108
-0.5925	0.7767	-0.5144
-0.0709	-0.2816	0.0171
0.8338	-0.4769	-0.3653
0.9011	-0.2749	-0.2171
0.5011	0.5108	0.1458
0.6087	-0.7953	-0.3160
1.1356	0.7763	0.3048
-0.2411	-0.0945	0.0781
-0.8867	0.5636	-0.4286
0.5653	1.1776	0.1455
0.9458	-0.2954	-0.2369
-0.7981	0.5944	-0.4529
-1.1015	0.3321	-0.1999
-0.0615	0.4428	-0.1399
-0.9810	-1.1845	0.8967
0.3588	0.3691	0.0110
1.0929	1.0728	0.3501
-0.1094	0.2835	-0.1209

Discussion-

From the above weight matrix table, it is evident that this method does not produce any significant reduction in weight values. In the weight table, we findsome of the weights are reduced to minimum i.e. less than 0.001 and some of the weights are saturated around 1.5 to 2.5. The error surface is very much sensitive to the regularization parameter (lamda) and the free parameter w0. The increase in the value of lamda saturates the network and the network does not converge to desired minimum value. In some of the simulations, it is observed that

when the value of w0 is set to 1.1 the network saturates very quickly without any significant reduction in error. The point to be noted here that this method gives better results than the other two penalty functions and it saturates much faster than those methods.

Sensitivity Analysis Method-

In this method of pruning, the network is trained normally done. During the training process, in each epoch the weight sensitivities are calculated according to the formula given in chapter-3. After training procedure, the low sensitivity weights are deleted. In this method, the threshold sensitivity is determined in such a manner that after deletion of weight the increase in SSE is within certain range. The process is tried with both 30 and 40 hidden neuron architecture using sigmoid activation function in both output and hidden layer. The pruned weights are given in the table 4.68,4.69,4.70,4.71.

For architectures containing 20 hidden neurons
Threshold sensitivity for both input-hidden and
hidden-output layer weights =0.01

Table no 4.68 both layers containing sigmoid (beta=0.5)activation
Final SSE=0.0084

w1		w2
1.1120	1.3280	1.6650
1.4350	1.5691	2.2012
0	0	-0.7978
0	0	-0.1166
0	-1.1338	0.9037
0.9523	0	-0.6988
0	0	-0.2441
-1.5618	1.6984	-2.8714
1.0744	1.2479	1.5747
1.1941	0	-1.2799
-1.1226	1.1827	-1.8310
0	0	-0.6570
-2.0871	-1.8553	3.5977
0	-0.8933	-0.6344
1.0694	0	-1.4586
0	0	0.2648
-0.9481	-0.9088	1.2074
0	0	0.1794
1.2950	1.4361	2.0092
1.0087	-0.9326	-1.8049

Table no.4.69 sigmoid(beta=1.0) in hidden & linear activation in output
Final SSE =0.0146

w1		w2
0	1.0414	0.2902
1.0926	0.7936	0.3785
0.3620	0	-0.2236
0.5641	0.2533	-0.1075
0	-1.0609	0.1607
0.9219	0	-0.2030
-0.3517	0	-0.1620
-1.0563	0.8848	-0.5557
0.5757	0.8911	0.3649
1.0984	0	-0.3106
-0.8162	0.6621	-0.3674
0	0.3763	-0.2481
0	-1.1972	0.6524
0	-0.8787	-0.2054
0.8918	0	-0.4147
0	-1.1085	0.0142
-0.5759	-0.6450	0.2333
0	0	-0.0131
0.6416	0.9961	0.4535
0.3915	-0.4668	-0.5302

For architectures having 30 hidden neurons
Threshold sensitivity for input-hidden layer weights, Sih<0.005
Threshold sensitivity for hidden-output layer weights, Sho<0.1

Table no 4.70 both layers containing sigmoid (beta=0.5)activation Final SSE=0.1110	Table no.4.71 sigmoid(beta=1.0) in hidden & linear activation in output Final SSE =0.0146

w1		w2
0.4646	0	0
1.0551	-1.0429	-0.3521
0	0	0
-0.8922	0	0.1227
0	-0.4938	0
1.0695	-0.5047	-0.1982
-0.3289	0.6171	-0.1889
-0.4156	-0.9687	0.2190
0.6747	-1.0137	-0.2699
-0.4845	0.5274	-0.1494
-0.7749	0.6402	-0.2452
0.4512	0.3716	0
-0.5813	0.7586	-0.2950
0	-0.2936	-0.0579
0.8287	-0.4726	-0.2062
0.9114	0	-0.0737
0.4932	0.5067	0.1211
0.6123	-0.7958	-0.1760
1.1386	0	0.2273
0	0	0
-0.8866	0.5479	-0.2504
0	1.1843	0.1356
0.9553	0	-0.0833
-0.7953	0.5728	-0.2695
-1.0985	0	-0.1106
0	0.4377	-0.0937
-0.9785	-1.1829	0.3873
0	0	0
1.0899	1.0719	0.2488
0	0.2763	-0.0959

w1		w2
0	0	0
1.6402	-1.5650	-2.5705
0	0	0
-0.9768	0	0.9724
0	0	0.3185
1.2742	-0.9747	-1.4300
0	0.8225	-0.8941
-1.0629	-1.2717	1.8865
1.2193	-1.2696	-1.6660
-0.7174	0.7884	-0.9497
-1.1482	1.1712	-1.7097
0	0	0.4210
-1.0801	1.2041	-1.7080
0	0	0.2720
1.0440	-0.8124	-1.1105
0.9831	0	-0.6726
0.6588	0.6422	0.7676
0.9641	-1.0014	-1.1750
1.4582	1.3281	2.1350
0	0	0.2810
-1.1810	1.1271	-1.6427
1.0320	1.2846	1.4364
1.0366	0	-0.7602
-1.1311	1.1201	-1.6216
-1.1829	0.7917	-1.0701
0	0	-0.1956
-1.8138	-1.8617	3.6160
0	0	0.3656
1.5989	1.5515	2.5779
0	0	-0.1421

RESULTS OF ARCHITECTURES OF 30 HIDDEN NEURONS WITHOUT PRUNING

Table no.4.72 sigmoid(beta=1.0) in hidden & linear activation in output
Final SSE =0.0055

w1		w2
0.4626	0.1025	0.0457
1.0621	-1.0686	-0.6462
-0.0585	-0.0248	0.0578
-0.8835	-0.2335	0.3089
-0.0066	-0.4843	0.0531
1.0595	-0.5417	-0.4134
-0.3675	0.6292	-0.3230
-0.4389	-0.9534	0.5493
0.7052	-1.0274	-0.4659
-0.5046	0.5278	-0.3029
-0.8042	0.6535	-0.4916
0.4471	0.3672	0.0469
-0.6343	0.7793	-0.5378
-0.0710	-0.2812	0.0258
0.8328	-0.5110	-0.3792
0.8979	-0.2989	-0.2097
0.5118	0.5214	0.2177
0.6164	-0.8146	-0.3517
1.1396	0.7944	0.4248
-0.2393	-0.0943	0.0876
-0.9098	0.5757	-0.4678
0.5772	1.1788	0.2512
0.9418	-0.3217	-0.2310
-0.8255	0.6057	-0.4879
-1.1098	0.3403	-0.2412
-0.0728	0.4402	-0.1167
-0.9951	-1.1743	0.9078
0.3596	0.3698	0.0629
1.1010	1.0819	0.4820
-0.1170	0.2820	-0.1035

Table no 4.73 both layers containing sigmoid (beta=0.5)activation
Final SSE=0.0024

w1		w2
0.4694	0.1233	0.1871
1.5295	-1.4416	-2.2517
-0.0589	-0.0318	0.1602
-0.9483	-0.5130	0.8433
-0.0692	-0.5033	0.2871
1.2220	-0.8732	-1.2508
-0.5502	0.7786	-0.8015
-0.9303	-1.1933	1.6475
1.1182	-1.1954	-1.4472
-0.6605	0.7324	-0.8465
-1.0559	1.0696	-1 5004
0.4823	0.4044	0.3641
-0.9737	1.1150	-1.5046
-0.1057	-0.3020	0.2434
0.9941	-0.7364	-0.9786
0.9629	-0.4601	-0.5932
0.6192	0.6050	0.6733
0.8901	-0.9435	-1.0267
1.3728	1.2058	1.8501
-0.2496	-0.1254	0.2545
-1.1022	1.0203	-1.4350
0.9286	1.2432	1.2238
1.0136	-0.5063	-0.6686
-1.0466	1.0184	-1.4214
-1.1534	0.6998	-0.9207
-0.1137	0.4632	-0.1905
-1.6548	-1.7192	3.1786
0.3934	0.3944	0.3174
1.4865	1.4303	2.2393
-0.1377	0.3045	-0.1422

RESULTS OF ARCHITECTURES WITH 20 HIDDEN NEURONS WITHOUT PRUNING

Table no 4.74 both layers containing sigmoid (beta=0.5)activation
Final SSE=0.0024

w1		w2
1.1120	1.3280	1.6650
1.4350	1.5691	2.2012
0.4542	-0.2616	-0.7978
0.5564	0.1812	-0.1166
-0.7366	-1.1338	0.9037
0.9523	-0.2979	-0.6988
-0.3002	-0.0210	-0.2441
-1.5618	1.6984	-2.8714
1.0744	1.2479	1.5747
1.1941	-0.7432	-1.2799
-1.1226	1.1827	-1.8310
-0.3195	0.4430	-0.6570
-2.0871	-1.8553	3.5977
0.3466	-0.8933	-0.6344
1.0694	-0.8058	-1.4586
-0.2867	-1.1150	0.2648
-0.9481	-0.9088	1.2074
-0.3537	-0.4226	0.1794
1.2950	1.4361	2.0092
1.0087	-0.9326	-1.8049

Table no.4.75 sigmoid(beta=1.0) in hidden & linear activation in output
Final SSE =0.0055

w1		w2
0.6026	1.0438	0.6693
1.1028	0.8711	0.8348
0.3508	-0.0475	-0.4612
0.5454	0.1893	-0.1009
-0.4483	-1.0610	0.4033
0.9206	-0.0085	-0.3463
-0.3349	-0.0503	-0.2356
-1.0868	0.9753	-1.1401
0.6996	0.9485	0.6999
1.0964	-0.1080	-0.6103
-0.8590	0.7286	-0.8562
-0.2582	0.4027	-0 4591
-1.1130	-1.2117	1.3270
0.1189	-0.8763	-0.2918
0.9130	-0.3602	-0.7207
-0.2251	-1.1076	0.1248
-0.6252	-0.6862	0.5652
-0.3468	-0.4120	0.0420
0.7820	1.0444	0.8545
0.5348	-0.5826	-0.9774

XOR OUTPUT TABLE FOR DIFFERENT PATTERNS

Table 4.76 FOR ARCHITECTURES CONTAINING 20 HIDDEN NEURONS

input values	desired output	case1	case2	case 3	case 4	case 5	case 6	case 7	case 8	case 9	case10	case11	case12
-1 1	1	0.9960	0.9930	0.9899	0.9273	0.7743	0.9866	0.9859	0.9952	0.9949	0.9793	0.9899	0.9273
1 -1	1	0.9992	0.9891	0.9687	1.0241	0.7884	0.9846	0.9728	0.9921	0.9919	0.8170	0.9687	1 0241
-1 -1	-1	-0.9916	-0.9960	-0.9862	-0.9734	-0.8259	-0.9839	-0 9820	-0.9951	-0.9955	-0.9597	-0.9862	-0.9734
1 1	-1	- 0.9975	-0.9971	-0.9789	-0.9929	-0.8159	-0.9804	-0 9720	-0.9909	-0.9912	-0.9058	-0 9789	-0.9929

where

case 1 = hidden layer having sigmoid(beta=0.5) and output layer having linear activation in Hessian matrix method

case 2 = both hidden layer & output layer having sigmoid(beta=0.5) activation in Hessian matrix method

case 3 = hidden & output layer having sigmoid(beta=0.5) activation & omega = 1.8 in Iterative pruning method

case 4 = hidden layer having sigmoid(beta=0.5),output layer having linear activation & omega=1.5 in Iterative pruning method

case 5 = hidden & output layer having sigmoid(beta=0.5) activation & lamda=0.1 in weight decay method-1

case 6 = hidden layer having sigmoid(beta=0.5),output layer having linear activation & lamda=1e-2 in weight decay method-1

case 7 = hidden & output layer having sigmoid(beta=0.5) activation & lamda=1e-3 in weight decay method-2

case 8 = hidden layer having sigmoid(beta=0.5),output layer having linear activation & lamda=1e-3 in weight decay method-2

case 9 = hidden layer having sigmoid(beta=0.5),output layer having linear activation & lamda=1e-4,w0=1.05 in weight elimination method

case 10 = hidden & output layer having sigmoid(beta=0.5) activation & lamda=1e-3,w0=0.9 in weight elimination method

case 11= hidden & output layer having sigmoid(beta=0.5) activation for simple convergence

case 12 = hidden layer having sigmoid(beta=0.5),output layer having linear activation for simple convergence

Table 4.77 FOR ARCHITECTURES HAVING 30 HIDDEN NEURONS

input values	desired output	case1	case 2	case 3	case 4	case 5	case 6	case 7	case 8	case 9	case 10	case 11	case 12
-1 1	1	1.0062	-1.0000	0.9782	0.9982	0.7434	0.9800	0.9970	1.0001	0.8715	0 8716	0 9782	0.9982
1 -1	1	0.9903	1.0000	0.9849	1.0184	0.7449	0.9801	0.9967	1.0001	0.8722	0.8729	0 9849	1.0184
-1 -1	-1	-1.0026	-0.9998	-0.9615	-0.9957	-0.7229	-0.9801	-0.9967	-0.9999	-0.9108	-0.8727	-0.9615	-0.9957
1 1	-1	-0.9831	-0.9997	-0.9860	-1.0083	-0 7288	-0.9823	-0.9964	-0.9999	-0 9955	-0.8816	-0.9860	-1.0083

where
case 1 = hidden layer having sigmoid(beta=1.0) and output layer having linear activation in Hessian matrix method
case 2 = both hidden layer & output layer having sigmoid(beta=0.5) activation in Hessian matrix method
case 3 = hidden & output layer having sigmoid(beta=0.5) activation & omega = 0.5 in Iterative pruning method
case 4 = hidden layer having sigmoid(beta=1.0),output layer having linear activation & omega=1.5 in Iterative pruning method
case 5 = hidden & output layer having sigmoid(beta=0.5) activation & lamda=0.8 in weight decay method-1
case 6 = hidden layer having sigmoid(beta=1.0),output layer having linear activation & lamda=0.01 in weight decay method-1
case 7 = hidden & output layer having sigmoid(beta=0.5) activation & lamda=1e-2 in weight decay method-2
case 8 = hidden layer having sigmoid(beta=1.0),output layer having linear activation & lamda=1e-2 in weight decay method-2
case 9 = hidden layer having sigmoid(beta=1.0),output layer having linear activation & lamda=1e-5,w0=1.1 in weight elimination method
case 10 = hidden & output layer having sigmoid(beta=0.5) activation & lamda=1e-3,w0=0.95 in weight elimination method
case 11= hidden & output layer having sigmoid(beta=0.5) activation for simple convergence
case 12 = hidden layer having sigmoid(beta=1.0),output layer having linear activation for simple convergence

CHAPTER-5
CONCLUDING REMARKS

In the thesis, back propagation algorithm is used to study the pruning algorithms. to make the back propagation algorithm more efficient for comparing the different penalty functions and algorithms, the following commonly observed characteristics should to taken into consideration -:

⇒ Good training procedure like CG(conjugate gradient methods), Levenberg & Marquardt method should be followed, which will make the ANN converge despite of all the complexities input patterns. This is an essential requirement to apply the pruning algorithms to the networks. The deletion of weights result in increase in error and it is the training procedure which at that time make the network converge again.

⇒ The learning of hidden units is influenced by learning process of the output units. Hence H.Takechi and K.Murakami (1994) proposed higher learning rate for the output layer than the hidden layer. This will result in the learning of some hidden units faster than others and hence, will result in an unbalanced influence on the hidden units on the network. This will be helpful in removing the hidden units, which have little influence. Even it was proposed to have different learning rates for all the weights, which will be, updated either in batch mode or pattern mode.

⇒ The initial weights should be chosen between $\left(\frac{-2.4}{F_i}, \frac{2.4}{F_i}\right)$, where F_i is the fan-in of the ith node for every layer. The learning process may be thought from the point of view of source, sink, vertex etc where the initial weights follow different paths to

reach the global minima. Hence though initialized within the accepted range, the convergence speed differs with different initialization of weights. Hence it is advisable to search the initial weight that will provide best converge and then apply all pruning methods and we will get networks whose performance can be compared.

⇒ For good convergence speed momentum term should be incorporated. It is also advisable to have different momentum term for different synaptic weights which will be changed according to their derivative w.r.t. the cost function of ANN.

CONCLUSION—

- The Hessian matrix method is found to be the best pruning method as it provides the minimum architecture. Hence it is advisable to use this method of pruning if we have a program for back propagation algorithm, as it will reduce the increase in error after pruning during the re-training process. The selection of the saliency factor for deletion of weights plays a crucial role as indicated by the results of XOR problem.The main disadvantage of this method is that it can not be applicable to non-differentiable activation functions.
- As iterative pruning method is a post-training method ,it is preferred when we do not have a program for back propagation algorithm for retraining to reduce the increase in error due to pruning. The pruning procedure does not depend on the training procedure and is controlled by only one parameter. This method does not provide a minimum architecture than what is produced in Hessian matrix method.
- The penalty function methods do not at all provides a good method for pruning. Although they are the only pruning methods, which are run simultaneously with training procedure. The selection of regularization parameter affects the convergence of the ANN and the results.
- Sensitivity method depends on the number of epochs. Hence the threshold weight sensitivity for deletion of weights depends on the training procedure.

AREAS FOR FUTURE RESEARCH—

- The areas like signal processing, vibration analysis etc., which involve analysis in frequency domain which are complex in nature can only be included into the domain of Neural networks through complex domain analysis of all real-valued architectures. That opens the search for the optimal network architecture, which can be solved by applying pruning algorithms to complex domain architectures.
- The basic practical application of neural networks is to predict events of future. Hence the scope of pruning algorithms can be extended to find the minimized architecture in those architectures such as FIR, BPT, RTRL networks.
- The penalty function methods do not suggest the general criterion for choosing a penalty function. Hence that could be an area of research.
- The Hessian matrix method provides an efficient procedure for pruning, but they are based on the approximation that the error surface will be quadratic in local or global minimum to initiate the pruning process. The effect of Hessian matrix method without that approximation should be investigated.

REFERENCES

1. Fahlman, S.E. and C. Lebiere(1990)---"*The cascade-correlation learning architecture*" in Advances in Neural Information Processing Systems 2 pp. 524-532
2. Lee, T. C., A.M. Peterson, and J.C. Tsai(1990)---"*A multi-layer feed-forward neural network with dynamically adjustable structure*". IEEE International Conference on Systems, Man and Cybernetics, pp 367-369
3. Werbos, P.J.(1974)—"*Beyond regression. New tool for prediction and analysis in the behavioral sciences*" . Ph.D. thesis, Harvard University
4. Rumelhart, D.E., G. E. Hinton, and R.J. Williams,(1986)—"*Learning representations by back-propagation errors.*" Nature(London) 323, pp 533-536
5. M.C.Mozer and P.Smolensky—"*Skeletionization· A technique for trimming the fat from a network via relevance assessment*" in Advances in Neural Information Processing I,D.S.Touretzky,Ed Morgan Kaufmann,1989,pp. 107-115
6. E.D.Karnin—"*A simple procedure for pruning back-propagation trained neural networks*", IEEE Transactions on Neural Networks, vol 1,no. 2,pp239-242, 1990
7. Y.Le Cun,J.S.Denkar,S.A.Solla—"*Optimal brain damage*" in Advances in Neural Information Processing(2),D.S.Touretzky, Ed(Denver1989),1990,pp 589-605
8. Y.Chauvin—"*Generalization performance of overtrained back-propagation networks*", in Neural Networks,Proc. EUROSIP Workshop,feb 1990,pp 46-55
9. MacQueen,J.B-"*Some Methods for Classification and Analysis of Multivariate Observations,*" Proc. 5th Berkeley Symposium on Mathematical Statistics and Probability,1967,pp 281-297
10. M. Ishikawa-" *A structural learning algorithm with forgetting of link weights*" , Tech. Rep. TR-90-7,Electrotechnical Lab., Japan,1990
11. A.N.Brukitt-"*Optimization of the architecture of feed-forward neural networks*"-Complex System, vol 5,pp 371-380,1991
12. S.Haykin –*"Neural Networks"*
13. M.H. Hassoun – *"Fundamental of Artificial Neural Networks*"-PHI,1999
14. M.K. Shrivastava – *"Dimensionality Reduction Techniques for Intelligent Systems"*-M. Tech. Thesis (March,1998)

DERIVATION OF BACK-PROPAGATION ALGORITHIM

GRADIENT-DESCENT METHOD

The method consist of two parts, one is forward pass and second one is learning phase. Let us consider the on-line training procedure where the weights are updated on pattern-by-pattern basis. The adjustments to the weights are made in accordance with the respective errors computed for each pattern presented to the ANN.

Forward pass:

Let the network consist of M layers and each layer has the weight matrix connected to its previous layer given by $W_{ij}^{(l)}$, where l=1,2, …. ,M . Here i is the node at the same layer and j is the node at the previous layer.

The net input to a node i in any layer l is given by

$net_i^{(l)} = \sum_{j=1}^{N_{l-1}} W_{ij}^{(l)} x_j^{(l-1)}$ where N_{l-1} indicates the number of nodes in (l-1) layer.

X_j indicates the input signals to l^{th} layer from (l-1) layer

Let the activation function of a neuron i in l^{th} layer is given by $f(net_i^l)$

Hence the output of the neuron i in l^{th} layer is given by

$x_i^l = f(net_i^l)$ where x_i^l is the output of the i th node of l th layer

Let the desired output of neuron j at output layer is given by d_j and the actual response is y_j

Hence the error signal at the output of neuron j at iteration n (i.e., presentation of the nth training pattern) is defined by

$$e_j(n) = d_j(n) - y_j(n)$$

The instantaneous sum of squared errors of the network is thus written as

$$E(n) = \frac{1}{2}\sum_{j \in C} e_j^2(n)$$

where the set C includes all the neurons in the output layer of the network

The sum of errors *E(n)* is a function of free parameters(i.e. synaptic weights and thresholds) of the network. For a given training set, *E* represents the cost function as the measure of training set learning performance.

Learning pass:

In this step of the algorithm, the change in weights (both synaptic and bias) are calculated and added to the original weights to update it. The change in weight is calculated on the basis of calculating the gradient of that weight w.r.t the error.

Let $w_{ji}(n)$ be a weight of any layer and $\Delta w_{ji}(n)$ is the corresponding change in weight at n^{th} iteration. Hence the change in weight is defined as

$\Delta w_{ji}(n) = -\eta \frac{\partial E(n)}{\partial w_{ji}(n)}$ where η is known as learning-rate parameter of back-propagation algorithm

The above equation can be rewritten as $\Delta w_{ji}(n) = \eta \delta_j(n) x_i(n)$

where local gradient $\delta_j(n)$ is defined as $\delta_j(n) = -\frac{\partial E(n)}{\partial w_{ji}(n)}$

Case-1 if neuron j is an output node

$\delta_j(n) = (d_j(n) - y_j(n)) f'(net_j)$

where $d_j(n)$ is desired output at j^{th} node at n^{th} pattern and $y_j(n)$ is corresponding actual output. *f(x)* represent the activation function of output layer neuron.

So the weight updating part is given by $\Delta w_{ji}(n) = \eta (d_j(n) - y_j(n)) f'(net_j) x_i(n)$

where $x_I(n)$ is output of I th node in the previous layer

Case-2 if neuron j is a hidden node in k th layer

$$\delta_j(n) = \left[\sum_{l=1}^{L} \delta_l w_{lj}\right] f'(net_j)$$

where *L* is no. of outputs in *k*+1 layer and

δ_l is local gradient of l^{th} neuron

So the weight updating part is given by $\Delta w_{ji}(n) = \eta \delta_j(n) x_i(n)$

Case –3 if the neuron *j* is the first hidden layer

Then the $x_i(n)$ are the training input patterns and the weight updating part is same as that of case – 2.

LEVENBERG-MARQUARDT METHOD-

To accelerate convergence speed in back-propagation algorithm, many methods are discovered e.g.

- Techniques such as varying the learning rate, using momentum and rescaling variables after each epoch.
- Techniques which takes the second order information into consideration such as conjugate gradient, quasi-Newton methods, Levenberg Marquardt method.

This Levenberg-Marquardt approach in nonlinear least squares applicable to batch training. This batch training indicates the updating of weights only once in an epoch, i.e. presentation all training patterns once.

Let the network in to learn association between the net of input-output pairs $\{(P_1,T_1),(P_2,T_2),\ldots\ldots(P_Q,T_Q)\}$

where $P_i = \left(p_{1_i}, p_{2_i}, \ldots, p_{m_i}\right)$ and m = number of inputs

$T_i = \left(t_{1_i}, t_{2_i}, \ldots, t_{n_i}\right)$ and n = number of outputs

For an M layer network,

the weight matrix W^i between layer i and i-1is given by
$W^i=[W^i(1,1),W^i(1,2),\ldots\ldots\ldots W^i(2,1),\ldots,W^I(k,j),\ldots]$

the bias matrix b^i between layer i and i-1 is given by
$b^i=[b^i(1), b^i(2),\ldots, b^i(k),\ldots]$

where the first index k denotes the node k of the ith layer and second index j denotes the node j of the (i-1)th layer

Now defining the combined weight matrix(**x**) of the total network as given as follows

$$\mathbf{x}=[W^1, b^1, W^2, b^2, \ldots\ldots W^M, b^M]$$

Let the performance index for the network is defined as $V = \frac{1}{2}\sum_{q=1}^{Q} e_q^T e_q$

where $e_q = \sum_{i=1}^{n}(T_{q_i} - A_{q_i})$ and A_{q_i} is the output of i^{th} node at presentation of q^{th} pattern

Let us define Jacobian matrix of the network as

$$J = \begin{bmatrix} \frac{\partial e_{11}}{\partial x_1} \frac{\partial e_{11}}{\partial x_2} \cdots \frac{\partial e_{12}}{\partial x_1} \frac{\partial e_{12}}{\partial x_2} \cdots \frac{\partial e_{1n}}{\partial x_1} \cdots \frac{\partial e_{1n}}{\partial x_N} \\ \frac{\partial e_{21}}{\partial x_1} \frac{\partial e_{21}}{\partial x_2} \cdots \frac{\partial e_{22}}{\partial x_1} \frac{\partial e_{22}}{\partial x_2} \cdots \frac{\partial e_{2n}}{\partial x_1} \cdots \frac{\partial e_{2n}}{\partial x_N} \\ \cdots\cdots\cdots\cdots \\ \cdots\cdots\cdots\cdots \\ \frac{\partial e_{Q1}}{\partial x_1} \frac{\partial e_{Q1}}{\partial x_2} \cdots \frac{\partial e_{Q2}}{\partial x_1} \frac{\partial e_{Q2}}{\partial x_2} \cdots \frac{\partial e_{Qn}}{\partial x_1} \cdots \frac{\partial e_{Qn}}{\partial x_N} \end{bmatrix}$$

where N = total number of elements of the matrix **x**
Q = total number of patterns of training set
n = total number of output neurons

According to Newton's method, the change in network's weight matrix **x** is given by

$$\Delta x = -[\nabla^2 V(x)]^{-1}(\nabla V(x))$$

where $\nabla V(x)$ is gradient of performance index $V(x)$
$\nabla^2 V(x)$ is its Hessian matrix

and it can be shown that $\nabla V(x) = J^T(x)\, e(x)$
$\nabla^2 V(x) = J^T(x).J(x)+S(x)$
and $S(x) = \Sigma e_i(x)\nabla^2 e_i(x)$

For Gauss-Newton method it is assumed $S(x) \approx 0$

Hence collecting all above conditions, we have the weight updating part as -

$$\Delta x = [J^T(x)\, J(x)]^{-1} J^T(x)\, e(x)$$

with Marquardt-Levenberg modification to Gauss-Newton method the weight updating part becomes

$$\Delta x = [J^T(x)\, J(x)+\mu I]^{-1}\, J^T(x)\, e(x)$$

The parameters μ in multiplied by some β wherever a step would result in an increased $V(x)$ and when a step reduces $V(x)$, μ is divided by β.

Hence the new weights will be

$$x_{new} = x_{old} + \Delta x$$

Hence summarizing, the Levenberg-Marquardt algorithm may be described as :

Step-1: Present all inputs to the network and compute the corresponding outputs and errors. Compute the sum of squares of errors over all inputs($V(x)$)

Step-2: Compute the Jacobian matrix($J(x)$)

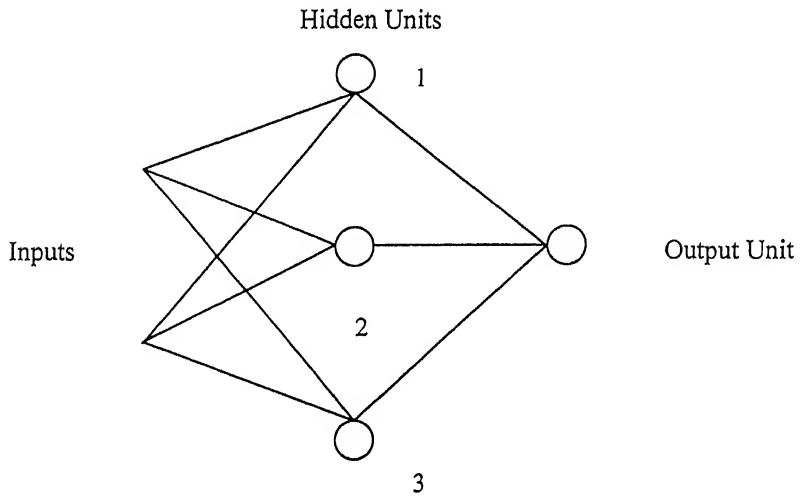
Step-3: Solve to obtain Δx.

Step-4: Re compute the sum of errors using $x+\Delta x$. If this new sum of squares is smaller than computed in step 1,then reduce μ by β ,let $x= x+\Delta x$ and go back to step 1.if the sum of squares is not reduced , then increase μ by β and go back to step 3.

Step-5: The algorithm is assumed to have converged when the norm of gradient $\nabla V(x)$ is less than some predetermined value, or when the sum of squares has been reduced to some specified error goal

DERIVATION OF ITERATIVE PRUNING FOR FFNN

The method is based on the idea of iteratively removing hidden units and then adjusting the remaining weights in order to maintain the original input-output behavior. In this method, the net input of the units fed by the unit to be removed is made approximately same after the removal of the unit adjusting all other inputs to that unit.



Let us consider a feed-forward architecture in which all the nodes of two layers are fully connected.

Let us consider unit 2 of the hidden layer. The receptive field for that hidden node is the all input nodes and the projective field for that hidden node in the output node. To remove hidden node 2, the net input to the output node is made approximately equal to the net input after the removal of the unit 2. This can be done by adding δ_w to the weights coming from unit 1 and 3, to the output node.

The receptive field of hidden unit 2 is the units, which feeds this unit. In this case, it is the input units. The projective field of the hidden unit 2 is the units, which are being fed by this unit. In this case it is the output unit.

Selection of the hidden unit to be removed :

Let us consider M training patterns.

Let $\overline{y}_i = \left(y_i^{(1)}.y_i^{(2)}.....y_i^{(M)}.\right)^T$

where y_i is the output of the unit on presentation of a training pattern.

and let the weight connection be denoted by w_{ji}

and we have $w_{ji}^2 \| \overline{y}_i \|_2^2$

the unit which gives the minimum value for the above term, in the unit to be removed.

So we have

$$h = \arg\min_{h \in V_H} \sum_{i \in \rho_h} w_{ji}^2 \| \bar{y}_i \|_2^2$$

hence the hidden unit h will be removed

Weight adjustment:

Let us define

$$y_{i,n} = [\bar{y}_{j1} \bar{y}_{j2} \ldots\ldots \bar{y}_{jn_{i-1}}]$$

where the entries are calculated over all training patterns and correspond to the units, which are in the respective field of the unit weights to which, is to be adjusted.

and $\bar{z}_{i,h} = w_{hi} \bar{y}_h$

so we have $Y_{i,h} \bar{\delta}_i = \bar{z}_{i,h}$ for every $i \in \rho_h$

i.e. for every node in the projective field of the hidden unit to be removed.

Hence we have $Y_{i,h} = diag\left(Y_{i_1,h}, Y_{i_2,h}, \ldots\ldots Y_{i_{\rho h},h}\right)$

$$\bar{\delta} = \left(\bar{\delta}_{i_1,h}^T, \bar{\delta}_{i_2,h}^T, \bar{\delta}_{i_{\rho h},h}^T\right.$$

$$\bar{z}_h = \left(\bar{z}_{i_1,h}^T, \bar{z}_{i_2,h}^T, \ldots\ldots\ldots \bar{z}_{i_{\rho h},h}^T\right)^T$$

and hence we have the system of equations defined as

$$Y_h \bar{\delta} = \bar{z}_h$$

This above equation can be solved by SVD, standard QR factorization method, but they are impractical for large and spare coefficient matrices. Hence the above equation can be very easily solved through iterative conjugate gradient (CG) methods with appropriate preconditioning method.

Let us define

k = number of columns of Y

D = (k x k) diagonal matrix where $D_{jj} = \| Y(:,j) \|_2^2$

L =(k x k) lower triangular matrix $L_{jk} = Y(:,j)^T Y(:,k), j > k$

C_w= preconditioning matrix = $(D + wL)D^{-1/2}$

where w= preconditioning parameter between (0,2) used to control the rate of convergence of the algorthim.

$\bar{\delta}_k$ is chosen to be null vector.

Hence we have the preconditioned normal system of equations

$$C_w^{-1} Y^T Y C_w^{-T} C_w^T \bar{\delta} = C_w^{-1} Y^T \bar{z}$$

The algorithm begins with the initial assumption of $\bar{\delta}$ and iteratively produces a sequence of points ($\overline{\delta k}$) in such a way that the residuals

$\rho(\overline{\delta k}) = \left\| \bar{z} - Y \overline{\delta k} \right\|_2^2$ are monotonically decreased

Hence summarizing, the iterative pruning algorithm may be stated as

Step 1: Identify the hidden unit to be removed
Step 2: Calculate the increase in the weights of the appropriate connections.
Step 3: Calculate the increased error after pruning
Step 4: Repeat the pruning process until the error after pruning is increased more than the acceptable limits.